DTIC copy

CROSS-RESOLUTION COMBAT MODEL CALIBRATION USING BOOTSTRAP SAMPLING

THESIS

Bryan S. Livergood, Captain, USAF

AFIT/GOR/ENS/98M-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/98M-16

CROSS-RESOLUTION COMBAT MODEL CALIBRATION USING BOOTSTRAP SAMPLING

THESIS

Bryan S. Livergood, Captain, USAF

AFIT/GOR/ENS/98M-16

THESIS APPROVAL

Name: Bryan S. Livergood, Capt, USAF

Class: GOR-98M

Thesis Title: Cross-Resolution Combat Model Calibration Using Bootstrap Sampling

Defense Date: 4 March 1998

Committee:	Name/Title/Department	Signature
Advisor	Kenneth W. Bauer, Jr. Professor of Operations Research Department Of Operational Sciences	Bauer
Reader	John O. Miller, Lt Col, USAF Assistant Professor of Operations Research Department of Operational Sciences	J. O. Miller
Reader	Raymond R. Hill, Major, USAF Assistant Professor of Operations Research Department of Operational Sciences	Raymond R Hill

Disclaimer

The views and opinions expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the U.S. Government.

CROSS-RESOLUTION COMBAT MODEL CALIBRATION
USING BOOTSTRAP SAMPLING

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Operations Research

Bryan S. Livergood, B.S.

Captain, USAF

March 1998

Acknowledgments

There are many people I would like to thank for their help in this research effort. First, I thank my advisor, Dr. Ken Bauer, for his virtually never-ending patience with me and for the way he continually guided me towards an attainable research goal instead of letting me wander off into the ether. I also thank my readers: Lt Col J. O. Miller, for his encouragement and sharp eye, and Maj Ray Hill for his help with BRAWLER and for introducing me to the bootstrap.

Thanks to Larry Taranto, Tammy Logan and the rest of the THUNDER and BRAWLER experts in ASC/XRA, whose help was invaluable in this research. Thanks also to Carol Johnson at RAND, who sent me a copy of the *enoughruns* program that she Tom Lucas wrote and that much of the work in this thesis is based on.

I also thank Maj Dave Davies and Lt Col J. J. Siegner, my partners on this SIMAF-sponsored research project; Maj Davies for, among other things, being a good TDY comrade, and Lt Col Siegner for leading the way in keeping our sponsor in the loop. Speaking of…, thanks to Terry Smith and his SIMAF team for providing us with a great research opportunity and outstanding support. Thanks also to Maj Dan Buschor who gave me all the help I needed with BRAWLER and let me use his scenarios, and whose blood, sweat, and tears went into getting BRAWLER up and running at AFIT. And thanks to Joel Coons for his computer support and conversation that made the time I spent in room 133b a little less painful.

Finally, I wish to thank my wonderful family. Thanks to my sister Nichole, whose newborn son Garren I just missed meeting when returning to school after Christmas break, for her continuing patience in waiting for me to meet my nephew. And thanks to my mom and dad, Shirley and Joe, for understanding why I hardly spent any time with them during their Thanksgiving 1997 visit, and why I frequently lost contact with them between January and March 1998. I love you all and appreciate all the support and encouragement you gave me during this wild ride.

Table of Contents

Page

List of Figures

Page

List of Tables

Page

AFIT/GOR/ENS/98M-16

Abstract

The US Air Force uses many combat simulation models to assist them in performing combat analyses. BRAWLER is a high-resolution air-to-air combat simulation model used for engagement-level analyses of few-on-few air combat. THUNDER is a low-resolution combat simulation model used for campaign-level analyses of theater-level warfare. BRAWLER is frequently used to ensure that THUNDER air-to-air inputs are valid. This thesis describes the confederation of THUNDER and BRAWLER by clearly showing how one particular BRAWLER output, the effectiveness of a missile type, is transformed into THUNDER air-to-air input data. Since BRAWLER is a stochastic simulation model, it is necessary to replicate a number of BRAWLER simulation runs in order to obtain a sufficiently accurate estimate of the mean missile effectiveness, a number that varies for each different BRAWLER combat scenario. This thesis focuses on using two different sequential methods to determine when the minimum number of BRAWLER runs has been performed to obtain a specified relative precision. One method uses classical statistical analysis techniques, while the other uses the more modern technique of bootstrap resampling. The performance of these two methods is compared.

CROSS-RESOLUTION COMBAT MODEL CALIBRATION USING BOOTSTRAP SAMPLING

1. Introduction

1.1. The Simulation and Analysis Facility

The Simulation and Analysis Facility (SIMAF) is a $7 million center currently being developed by the United States Air Force's (USAF) Aeronautical Systems Center (ASC). The SIMAF vision is to become the premiere facility for all ASC modeling, simulation, and analysis (MS&A) to support decisions concerning the acquisition of weapon systems at ASC. The SIMAF program is managed by ASC/SM. In May 1997, ASC/SM requested that the Air Force Institute of Technology (AFIT) provide them with technical guidance and analysis on a pilot effort to demonstrate SIMAF's capability. In particular, SIMAF wanted AFIT to help produce a proof-of-concept scenario demonstrating the utility of simulation environments in the acquisition cycle.

The SIMAF operational concept is that the center will be a "virtual facility" as well as a "physical facility." As a virtual facility, it will function as an integrator of products, tools, and technology and as a communications and networking node. As a physical facility, it will function as a physical gateway for MS&A to a synthetic battlespace environment, provide a common environment for both constructive and virtual analysis, and support multiple weapon acquisition programs at different levels of security (ASC/SM, 1997).

1.2. SIMAF Proof of Concept

To demonstrate proof of the SIMAF concept, a pilot program was created to demonstrate the so-called cross-resolution modeling process; specifically, the confederation of an engagement-level simulation and a campaign-level simulation. The campaign-level model used in this program is THUNDER, and the engagement-level model is BRAWLER. THUNDER is the USAF's primary campaign analysis tool, and BRAWLER is its primary air-to-air engagement analysis tool. Both THUNDER and BRAWLER are considered legacy models. BRAWLER became operational in 1976, while THUNDER's first production release was ten years later in 1986. While the current trend in modeling and simulation is toward virtual models, both of these legacy models are constructive models. These two particular models were chosen for three reasons:

1. There is a wealth of information available about each.
2. There are many expert users of the models available for consultation.
3. Both models are part of the USAF Analysis Toolkit.

As part of the USAF Analysis Toolkit, both will continue in use while the USAF transitions from current legacy models to new full-dimensional models (Langbehn, 1998).

An effort was made to include a virtual engagement-level model in this research, namely the USAF's second-generation version of the Man-in-the-Loop Advanced Air-to-Air System Performance Evaluation Model (MIL-AASPEM II). However, the newness of this model precluded its use, since at the beginning of this research undertaking neither a production version of the model nor appropriate training was available.

1.3. Cross-Resolution Modeling

This research demonstrates the confederation of THUNDER and BRAWLER. BRAWLER is a high-resolution model and THUNDER is a low-resolution model. THUNDER models many other aspects of warfare besides air combat, so in order to avoid being a prohibitively large model, its air-to-air model is less detailed than BRAWLER's. Their confederation is important because BRAWLER is used to ensure that THUNDER air-to-air inputs are valid, or in other words, to calibrate THUNDER. We say that THUNDER input data that has been properly calibrated by BRAWLER has a pedigree.

This research clearly shows how a particular BRAWLER output metric, called a measure of effectiveness (MOE), is transformed into input data for THUNDER. The specific BRAWLER MOE used in this research is a measure of missile effectiveness called adjusted kills per firing (AKPF). The AKPF MOE is calculated from BRAWLER missile effectiveness data using a process introduced in Section 3.4.5.

When we calibrate THUNDER using BRAWLER, we might say that this missile effectiveness data crosses a level of model resolution when passing from BRAWLER to THUNDER. By using this calibration process, we follow an important principle of cross-resolution modeling is, which is to ensure that a pedigree for the data is established before crossing a level of resolution. In this calibration process, THUNDER input data has a direct lineage to BRAWLER output data, and is therefore called a pedigree database.

1.4. Stochastic Simulation

In the current THUNDER calibration process, the AKPF MOE from BRAWLER is used to calibrate THUNDER input for air-to-air probability of kill (PK), which deals with the likelihood that an aircraft with a given weapon will kill its target. However, the AKPF may vary drastically for each realization of a BRAWLER engagement. This is because BRAWLER is a stochastic simulation model involving many random variables. This requires that BRAWLER engagements be replicated multiple times. The mean of the AKPF values obtained from the replications of an engagement is our best estimate of the expected value of AKPF over the long run, which we call the expected AKPF. To determine if our observed mean AKPF is sufficiently accurate for this purpose, a standard confidence interval is often calculated. If a sufficiently narrow confidence interval about the mean is obtained, we are confident in our estimate of the true AKPF is a good one. Therefore, we are confident of providing THUNDER with a good PK input value.

The number of replications required to obtain a sufficiently accurate estimate of the true AKPF varies for each BRAWLER scenario. A BRAWLER scenario is defined by information such as the types and numbers of aircraft involved in combat and the length of the simulation run. Different combinations of the numbers of aircraft involved in combat are referred to as engagement force structures. An example of an engagement force structure is four friendly aircraft versus four hostile aircraft (4v4). To obtain an accurate estimate of the true AKPF, we must ensure that this mean is calculated from a number of BRAWLER replications that is greater than or equal to *N*, where *N* is the number of replications necessary to meet a precision criterion. One such criterion that is often used by BRAWLER users to estimate means of MOE's and used in this thesis is the follows:

- Enough replications have been performed when the endpoints of a $100 \cdot (1-\alpha)\%$ standard confidence interval about the estimated mean are within $\pm\gamma\%$ of the estimated mean.

Here α is the significance level. Another way of stating this criterion is that the interval satisfies the requirement that the relative error ε of the estimated mean is equal to γ. This interval about the estimated mean is an approximate $100 \cdot (1-\alpha)\%$ confidence interval for the expected AKPF. We may also consider this precision criterion as a stopping criterion for performing BRAWLER replications.

1.5. The Campaign Analysis Group

One USAF analysis group that uses BRAWLER and THUNDER is the Aeronautical Systems Center Development Planning Campaign Analysis Group (ASC/XRA), who are a major potential user of SIMAF. ASC/XRA performs campaign-level analyses for the USAF's top-priority weapon system programs such as the Joint Strike Fighter (JSF) program. The JSF is currently being developed to replace the F-16 and perhaps other aircraft as well. Over the past several years, ASC/XRA has performed many BRAWLER replications of many different scenarios and has calculated $100 \cdot (1-\alpha)\%$ standard confidence intervals for many different combinations of the following:

1. the number of BRAWLER replications performed,
2. the value of the significance level α, and
3. the type of BRAWLER scenario that is performed.

These confidence intervals were used to help estimate how many BRAWLER replications of a scenario are actually necessary to meet their precision criterion, and these values are currently used as guidelines for similar scenarios. BRAWLER users in ASC/XRA use these guidelines as well as their own experience to choose the number of replications they wish to perform. For many 4v4 engagements, ASC/XRA typically performs between 200 and 300 BRAWLER replications (Taranto, 1998).

This information brought the following questions to mind:

1. Could another approximate $(1-\alpha)\%$ confidence interval besides the standard approximate confidence interval be calculated such that the stopping criterion is met with fewer replications?
2. If so, how would this interval compare with the standard confidence interval in its coverage of the expected AKPF?

1.6. Bootstrap Sampling

It turns out that one such method for calculating such an interval involves using a technique called bootstrap sampling, introduced in Section 4.5. The bootstrap sampling method involves analyzing pseudo-data obtained by resampling, or "bootstrapping", the original data a large number *B* of times. Bootstrapping is a technique recommended by Efron (1982) for constructing approximate confidence intervals. Using bootstrap sampling, we can calculate a bootstrap percentile interval. A $(1-\alpha)\%$ bootstrap percentile interval is another approximate $(1-\alpha)\%$ approximate confidence interval for the expected AKPF, and has subtle differences from a standard confidence interval.

1.7. Purpose of Research

The main purpose of this research is to investigate if the number of BRAWLER replications necessary to obtain a $(1-\alpha)\%$ bootstrap percentile interval that meets the stopping criterion above is less than the number required to obtain a $(1-\alpha)\%$ standard confidence interval that meets the same criterion, where in both cases $\alpha = \gamma = .10$. In this investigation, 200 BRAWLER replications were performed for each of three different scenarios, which are described in Section 4.2.4. The specific objective of the investigation can be described in four steps to complete for each scenario, as follows:

1. Calculate the estimated mean AKPF and a standard 90% confidence interval about the estimated mean using all 200 BRAWLER replications. Use this mean and confidence interval as our best estimate of the expected value of AKPF. We call this estimate our *Truth model*, since we are using it to represent the true expected AKPF.

2. Determine the minimum number N_1 of BRAWLER replications necessary to provide an approximate 90% standard confidence interval such that the relative error of the estimated mean is equal to .10. Use this estimated mean as another estimate of the expected value of AKPF. A sequential procedure that accomplishes this, denoted Candidate S1, is described in Chapter 4.6.2.

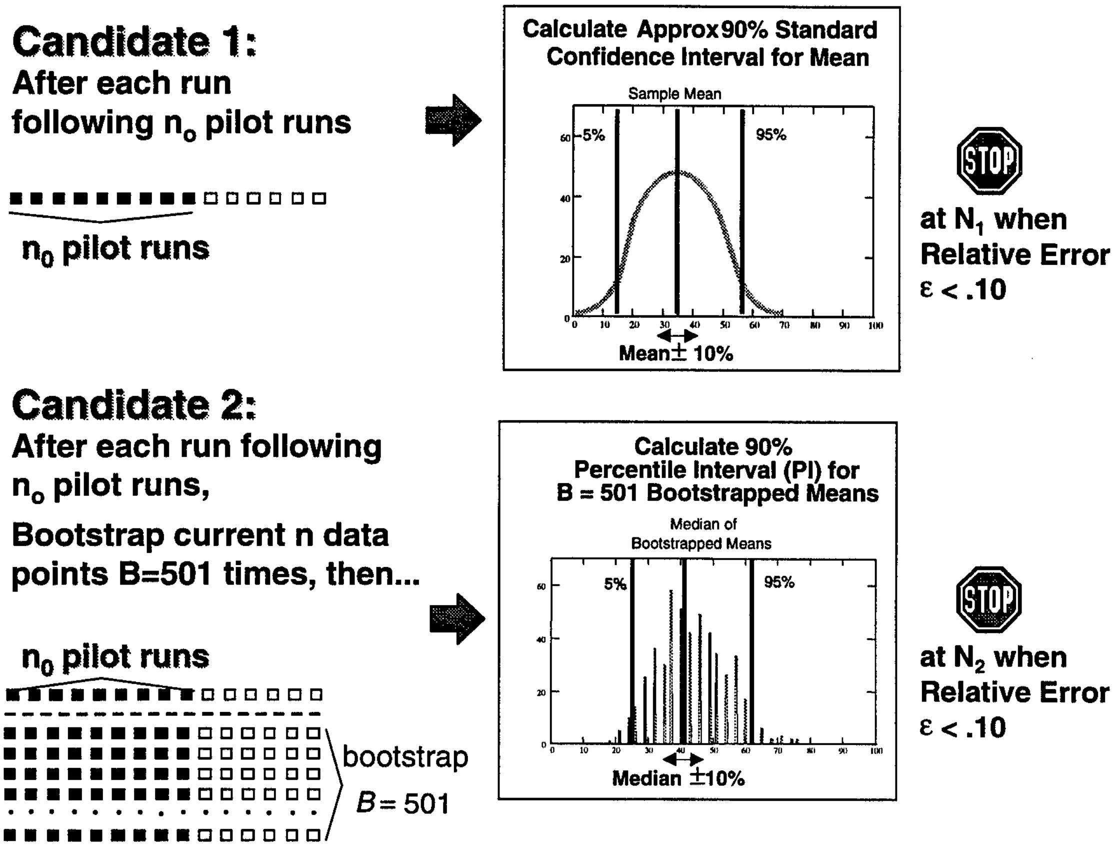
3. By bootstrapping the AKPF data B times, where B = 501, determine the minimum number N_2 of BRAWLER replications of each scenario necessary to provide a bootstrap 90% percentile interval that meets the same criterion as in step 2. Use this estimated mean as yet another estimate of the expected value of AKPF. A sequential procedure that accomplishes this, denoted Candidate S2, is described in Chapter 4.6.4.

4. Compare N_1 and N_2 and the two intervals generated in steps 2 and 3. Answer the following questions:

   a. Is the expected value of N_2 less than the expected value of N_1 (denoted $\mathrm{E}(N_2) < \mathrm{E}(N_1)$)? In other words, can 90% bootstrap percentile intervals using N_2 replications, where $N_2 < N_1$, be consistently constructed that meet the same stopping criterion as a standard 90% confidence interval using N_1 replications?

b. Do either (or both) of the two intervals calculated in steps 2 and 3 overlap the interval calculated in step 1? In other words, are the corresponding estimates of AKPF statistically similar?

The procedures used in Candidates S1 and S2 are displayed in Figure 1.1.



Candidate 1:
After each run following n_o pilot runs
n_0 pilot runs
Calculate Approx90% Standard Confidence Interval for Mean
Sample Mean
5%
95%
Mean± 10%
STOP
at N_1 when Relative Error ε < .10
Candidate 2:
After each run following n_o pilot runs,
Bootstrap current n data points B=501 times, then...
n_0 pilot runs
bootstrap
B= 501
Calculate 90% Percentile Interval (PI) for B = 501 Bootstrapped Means
Median of Bootstrapped Means
5%
95%
Median ±10%
STOP
at N_2 when Relative Error ε < .10

Figure 1.1. Procedures for Candidates 1 and 2

1.8. Thesis Format

Chapter 2 contains the results of a review of the literature on several different subjects related to this thesis. Chapter 3 provides a background discussion on the areas that are explored in this thesis. Chapter 4 discusses the methodology used to obtain the results of this research, and Chapter 5 presents those results. Chapter 6 summarizes the research, discusses the results, and presents ideas for additional research in related areas.

Supporting material for this thesis are contained in the appendices. Appendix A contains a *Mathcad 6.0* program written by the author and designed to run on a Windows™ personal computer (PC) called *bootstrap.mcd* that performs bootstrapping functions and contains algorithms that determine the number of BRAWLER replications required to accurately estimate mean AKPF. Appendix B contains a *FORTRAN 77* program written by Johnson and Lucas (1994) called *enoughruns.F* that is designed to run between BRAWLER replications on a Sun or Silicon Graphics workstation. It uses bootstrapping to determine if enough BRAWLER replications have been performed to accurately estimate the expected value of an MOE called Exchange Ratio. Appendix C contains a BRAWLER post-processing *AWK* program called *STATS* written by Taranto (1997). *STATS* calculates many BRAWLER MOE's using BRAWLER replication log files and generates an output file called *STATS.prt* that summarizes these MOE's. Example missile effectiveness reports from *STATS.prt* appear in Appendix D. Appendix E contains an *AWK* program called *msl_summary.awk,* a subprogram of *STATS*, in which the AKPF MOE is calculated. Appendix F contains a *Unix C-Shell* program written by the author called *make_flyAKPF* that runs *STATS* on each individual BRAWLER replication log file and creates one file of AKPF data that may be used for bootstrap sampling. Appendix G contains the BRAWLER AKPF and Firings data from 200 replications of each of five different BRAWLER scenarios. Finally, Appendix H presents the derivation of an equation for AKPF.

2. Literature Review

2.1. BRAWLER

Johnson (1994) investigated how many BRAWLER replications were required to obtain a relative error of .10 for a BRAWLER measure of outcome (MOO) called exchange ratio. Exchange ratio is defined as the number of Red aircraft killed (*RK*) in an engagement divided by the number of Blue aircraft killed (*BK*). The exchange ratio for a single BRAWLER replication *i* is expressed as follows:

$$ER_i = \frac{RK_i}{BK_i} \tag{2.1}$$

However, since BRAWLER is a stochastic simulation model (see Section 3.2.3 for details), we are more interested in the mean exchange ratio (*ER*) across *N* BRAWLER replications. *ER* is calculated as the sum of all Red aircraft killed (*RK*) divided by the sum of Blue aircraft killed (*BK*), defined as follows:

$$ER = \frac{\sum_{i=1}^{N} RK_i}{\sum_{i=1}^{N} BK_i} \tag{2.2}$$

where *N* is the total number of BRAWLER replications that have been performed.

Johnson found that the number of replications needed varies dramatically, depending on the number of aircraft in the engagement and the resulting *ER*. For example, a 4v4 engagement with a Blue PK = 1.00 and Red PK = 0.25, 117 runs were required, resulting in *ER* = 2.49. A similar 2v2 engagement required approximately 400 runs resulting in an approximate *ER* of 8.5.

Taranto (1995) describes a process for calibrating THUNDER with BRAWLER developed by ASC/XRA. This process, along with a pedigree database, as defined in Section 1.3, was developed for the Department of Defense's (DOD's) Joint Advanced Strike Technology (JAST) program, which evolved into the JSF program. In this calibration process, the performance of an individual aircraft/weapon combination versus an opponent aircraft is used to calibrate THUNDER air-to-air engagements with BRAWLER engagements.

2.2. THUNDER

Webb (1995) examined THUNDER's variability using a univariate analysis. He also examined the model's interrelationships using a multivariate analysis (principal components analysis and factor analysis). Finally, he examined THUNDER's sensitivity to input parameters.

Farmer (1996) modified the appropriate THUNDER input variables and used Design of Experiments (DOE) and Response Surface Methodology (RSM) to quickly show the effects of changing force structure. He also used factor analysis to find relationships among different MOE's to create new, simpler variables. The methods he used provide accurate, "quick turn" analysis tools for a force structure decision-maker to use.

ASC/XRA (1996) ran several experiments using THUNDER and identified four primary inputs that determine the outcome of THUNDER air-to-air engagements. They determined that it is possible to correlate two of these four inputs with BRAWLER results. This resulted in development of the methodology for THUNDER/BRAWLER calibration mentioned in Section 2.1. In the current calibration process, the BRAWLER MOE of adjusted kills per firing (AKPF) for a given type of missile in an engagement becomes the THUNDER air-to-air probability of kill (PK) input value.

Grier (1997) developed another "quick turn" tool designed to identify the cost and capabilities of alternative force structures using factor analysis and response surface methodology. Using these techniques, he was able to successfully "link procurement dollars to an alternative force structure's combat capability." His work allows the USAF to evaluate, within 48 hours, various force structures' combat capability in terms of theater level campaign objectives.

2.3. Cross-Resolution Modeling

Davis (1995) describes some of the challenges in connecting models that were developed independently. He recommends that connection activities for these types of models be guided by design work to identify how the models would have been developed if designed with integration in mind. He claims that this approach makes it possible to connect the models usefully and comprehensively.

Hillestad, Owen, and Blumenthal (1995) examined differences in outcomes predicted by different-resolution models using identical combat situations, which provided insights into the problems of aggregation. They found that intuition regarding outcomes, causes, and effects were frequently wrong, leading to bad aggregate approximations. They note that although scaling for different levels of resolution is possible, they were unable to find a method of predicting the appropriate resolution level scaling technique and factors in advance.

Hillestad and Moore (1996) investigated and implemented alternatives to different modeling aspects they believe are essential to improving theater-level campaign analysis. These aspects include how to create more flexible structures to simulate the wide range of future scenarios and their uncertainties, and how to link low-resolution models to high-resolution models in an analytically valid way. They built the Theater-Level Campaign (TLC) model to improve their capability to perform analysis in the future.

2.4. Bootstrap Sampling Method

Efron (1979) is the author of the first largely read introduction to the bootstrap and is often credited with popularizing the bootstrap as a way to estimate a sampling distribution. Efron (1982) presents more details about the bootstrap and its relationship to more well known methods such as the jackknife. Efron and Tibshirani (1993) offer a comprehensive review of bootstrap resampling techniques.

Johnson and Lucas (1995) developed a *FORTRAN 77* program called *enoughruns*, appearing in Appendix B, that uses the bootstrap percentile interval method, introduced in Section 4.5.3.4. The program *enoughruns* is focused specifically on the exchange ratio MOO and can be automatically called after completing 20 BRAWLER replications, or "runs." It determines if enough BRAWLER runs have been performed to satisfy a specific stopping criterion discussed in Section 4.6.3. If so, BRAWLER replications cease; if not, another BRAWLER replication is performed and the process is repeated after the next BRAWLER replication is complete. The stopping criterion is based upon the use of a bootstrap percentile interval, introduced in Section 4.5.3.3. Johnson and Lucas' methodology is discussed in detail in Section 4.6.3

Kim (1996) used bootstrap techniques in his work on the C-17 aircraft's personnel airdrop problem. He developed cumulative distribution functions of the maximum parachute centerline entanglement risk for the C-17 using bootstrap techniques, and then compared the effects of different C-17 aircraft configurations on the entanglement distribution function. Using these techniques, he showed that certain configurations of the C-17 showed a lesser entanglement risk than did the older C-141 aircraft.

DiCiccio and Efron (1996) survey bootstrap methods for producing approximate confidence intervals, with the goal of improving upon the accuracy of standard confidence intervals by an order of magnitude. They also provide an overview of four different bootstrap confidence interval procedures.

3. Background

3.1. Introduction

In this section we provide a background of topics pertinent to this research. We start with basic modeling and simulation concepts in Section 3.2. In Section 3.3 we discuss cross-resolution modeling and some of the issues involved with it. Finally, in Section 3.4 we discuss the combat models used in this research, BRAWLER and THUNDER.

3.2. Modeling and Simulation

3.2.1. Modeling and Simulation Definitions

The DOD defines constructive models as "a class of models or simulations executed by computer software without man-in-the-loop," and virtual models or simulations as "a class of models or simulations that feature live personnel (man-in-the-loop) in a computerized simulation environment." It states that the analysis of engagement-level models is "concerned with estimating the effectiveness of systems within various classes of engagements: air-to-air, surface-to-air, and air-to-surface with stated levels of performance," and that "the resulting estimates are called Measures of Effectiveness (MOE's)." It further states that the analysis of campaign-level models is "concerned with the cumulative long-term effects of kills and losses on the outcome of theater level conflict of campaign duration," and that it is "used to determine how forces should be used to achieve campaign objectives." It adds that "outcomes, measured in units that bear some relationship to the overall objectives specified in the planning scenarios, are called Measures of Outcome (MOO's)" (ASC/XRB, 1996).

Constructive simulation models are commonly used for engagement-level simulations and especially for theater-level simulations. Virtual models are not currently used at the theater level, due to their complexity, but are used effectively at lower levels such as the engagement level. Both constructive and virtual simulation models are common for engagement-level simulations, with the current trend towards virtual simulation.

3.2.2. Simulation Model Resolution

Campaign-level simulation models are not normally operated at the entity level; rather, they are usually operated using aggregate groups of entities to represent a single object. They are usually self-contained and may be very detailed in their particular areas of interest, but tend to be nominal in most areas (Hardy and Healy, 1994). For this reason, campaign-level models are usually considered low-resolution models.

Engagement-level simulations are usually modeled based on individual entities. Because they are modeled this way, they are usually far more detailed than campaign-level simulations and are usually considered high-resolution models. Because of differences in model levels and resolutions, it is useful to use a set of models to analyze complex processes such as warfare. The core model set that ASC/XRA uses and the hierarchy of these models is displayed in Figure 3.1. Recall that the models we are focusing on in this research are BRAWLER and THUNDER.

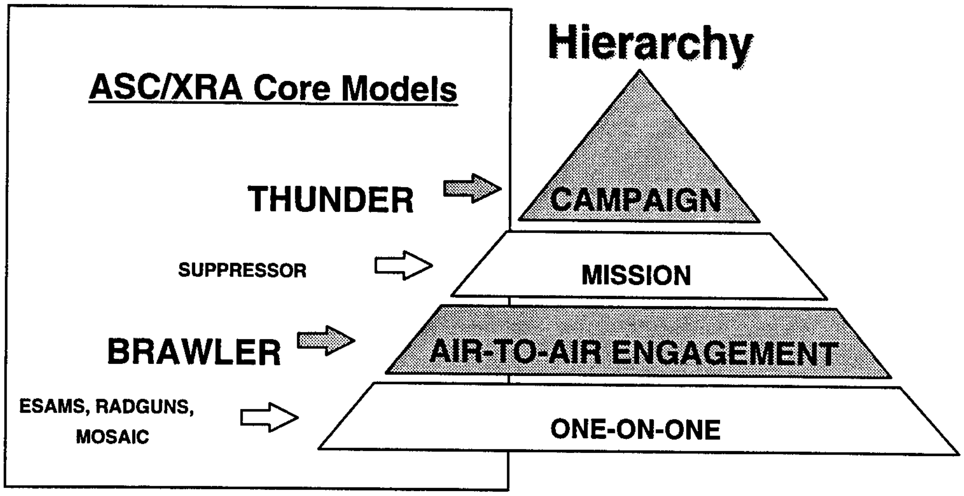


Figure 3.1. ASC Core Model Set (Taranto, 1995)

The main difference between the methodologies of THUNDER and BRAWLER is the level of detail in which they model air warfare. Whereas air-to-air attrition is very detailed in BRAWLER, in THUNDER it is modeled at a very aggregated level. For example, in THUNDER, the average effectiveness of a missile system versus a target aircraft is approximately the same regardless of whether or not the missile is the primary munition, is typically employed with specific tactics, or encounters the same threat system with different weapons. In contrast, weapons for both Red and Blue are explicitly modeled in BRAWLER and their performance is verified against flight tests and laboratory experiments. Tactics are modeled explicitly and are based upon intelligence estimates of threat weapon system capabilities, threat cultural biases and pilot proficiency (ASC/XRA, 1996).

3.2.3. Stochastic Simulation Models

Both BRAWLER and THUNDER are *stochastic* simulation models since they involve many random variables. Each realization of a BRAWLER engagement or a

THUNDER campaign is called a. Each BRAWLER and THUNDER output must be considered a random variable, the value of which varies from one replication to the next and follows a specific statistical distribution. Therefore a statistical analysis of the output data is required, requiring many BRAWLER and THUNDER replications. As mentioned in Section 1.4, the AKPF may vary drastically for each replication of a BRAWLER engagement.

A question with stochastic models is "how many replications are enough?" When trying to obtain a specified precision for a statistic such as the mean AKPF, the answer to this question is not obvious. It varies for each simulation model, and even for different statistics of interest within the same model. Law and Kelton (1991) present a sequential procedure for obtaining an estimate of the mean with a specified relative error and confidence level that allows the simulation user to only perform as many replications as necessary. This procedure was used in this research and is discussed in Section 4.6.1. Johnson and Lucas (1994) present a different sequential procedure that uses the bootstrap method, also used in this research and discussed in Section 4.6.3. The latter procedure is contained in the program *enoughruns*, previously mentioned in Sections 2.1 and 2.4.

3.3. Cross-Resolution Modeling

In campaign-level models such as THUNDER, much of the force-effectiveness input data such as attrition data comes from higher resolution models. The linking of high-resolution models to low-resolution models is called *cross-resolution modeling.* When specific models are linked in this manner, they are *cross-coupled.* When models of different levels are linked together, they are *vertically integrated.* Cross-resolution modeling and vertical integration is essential for performing quality campaign-level analyses; Hillestad and Moore (1996) list several reasons why, including:

1. Weapons system test results are usually provided with too much resolution for a campaign model and may cover only a small subset of the cases encountered in a theater campaign

2. Some phenomena must be modeled at higher levels of resolution to be represented judiciously in theater models.

3. It is difficult to compare the effects of weapon systems on campaign outcomes unless higher-resolution data about those systems are available.

4. Detailed models are needed to evaluate new systems whose technological characteristics can be defined but for which test data are not available.

Before beginning the cross-coupling process, one must first identify the high-resolution model(s) to use to provide attrition data to the low-resolution campaign model. BRAWLER is the air-to-air combat model that the USAF uses to provide air-to-air combat data to THUNDER. In other words, THUNDER is cross-coupled with BRAWLER with respect to air-to-air engagements. We may also say that THUNDER and BRAWLER are vertically integrated. We discuss in Section 3.4.5 how we calibrate the air-to-air aspects of the higher-level THUNDER model with the lower-level BRAWLER model using cross-resolution modeling techniques. BRAWLER may also be calibrated with still-lower-level engineering models that perform specific aircraft and weapon functions in much greater detail than BRAWLER does.

The cross-coupling process begins by generating a limited set of inputs over a range of situations that are likely to be encountered in the campaign-level model. Each of these situations may be coupled with a high-resolution rendition from the engagement-level model. Next, each situation in the campaign model is linked to one of the input cases. The mean outcome of THUNDER engagements is calibrated to approximate the mean outcome of similar engagements in BRAWLER. A scaling process allows any engagement whose outcome has not been decided, or *adjudicated*, in the high-resolution model to be derived from another engagement that was. THUNDER uses a sequential scaling process that takes combat between individual aircraft types and aggregates those results up to the flight group versus flight group level for an overall engagement loss score (AFSAA, 1996). This process is discussed in Section 3.4.4. Finally, the campaign model uses data from the engagement-level model in order to estimate attrition. (Hillestad and Moore, 1996)

3.4. Combat Models

3.4.1. Introduction

In this section, we provide information about the combat models used in this research. An overview of BRAWLER and THUNDER is provided in Sections 3.4.1 and 3.4.2, respectively. THUNDER's air-to-air submodel is described in Section 3.4.4. Finally, an analysis of the options considered by ASC/XRA for calibrating THUNDER with BRAWLER is presented in Section 3.4.5.

3.4.2. BRAWLER Overview

BRAWLER is a high-resolution air-to-air combat simulation model that is used for engagement-level analysis of few-on-few air combat. It is an event-driven, Monte Carlo simulation of within-visual-range and beyond-visual-range air-to-air combat engagements between multiple flights of aircraft, which are explicitly modeled through all phases of air-to-air combat. BRAWLER is considered a high-resolution model due to the engineering-level models of hardware and physical effects that it incorporates.

Typical uses of BRAWLER include aircraft performance trade studies, avionics and weapons effectiveness studies, and tactics development (ASC/XRA, 1996). BRAWLER is predominantly written in *FORTRAN*, with specific routines written in *C*.

Each run of BRAWLER simulates a single air-to-air combat engagement. Due to the stochastic nature of BRAWLER events, multiple replications of BRAWLER using the same initial starting conditions may yield drastically different results. BRAWLER produces results similar to more detailed exercises like flight tests and man-in-the-loop simulations. Figure 3.2 gives an overview of BRAWLER.

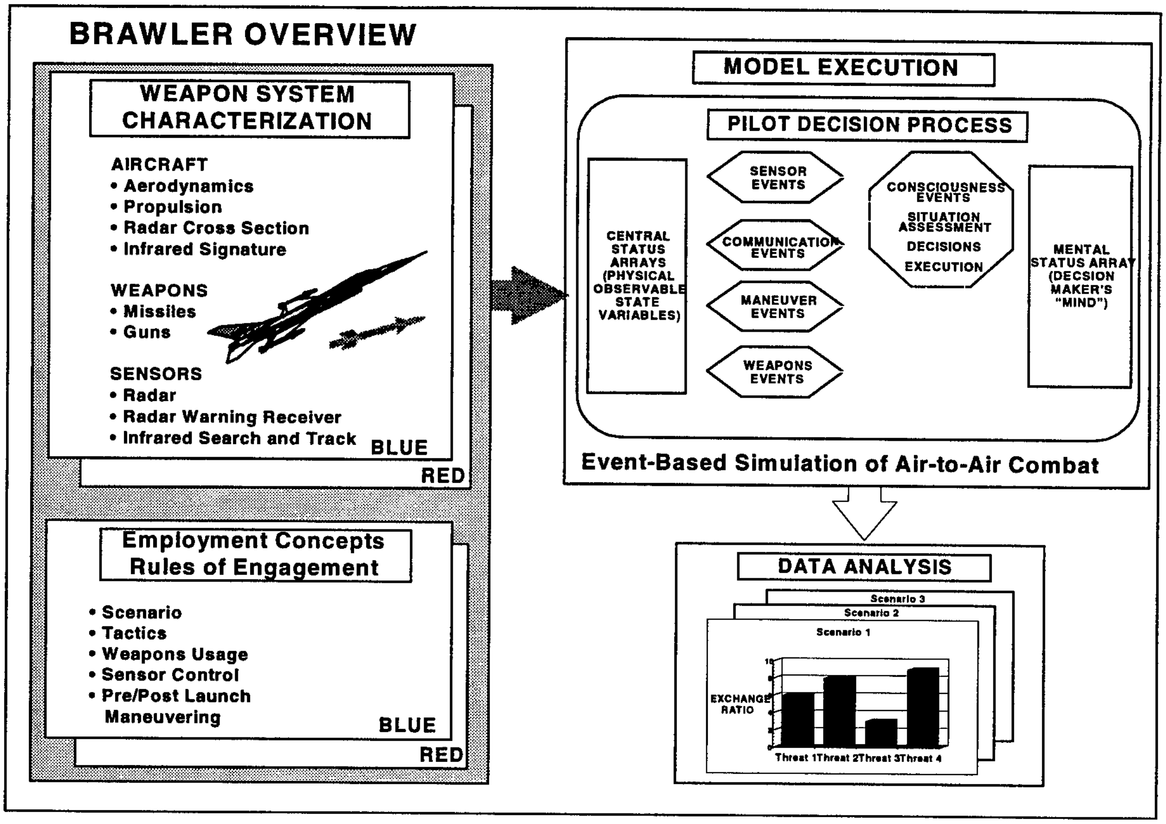


Figure 3.2. BRAWLER Overview (ASC/XRA, 1996)

3.4.3. THUNDER Overview

THUNDER is a two-sided theater-level stochastic simulation model of conventional air, land, and naval air warfare. THUNDER is written in the *SIMSCRIPT II.5*^{TM} programming language and is used by a large number of US and allied defense organizations and contractors. It is primarily used in order to evaluate force structures, conduct Analyses of Alternatives (AOA's), and develop strategies and tactics.

THUNDER models air warfare at a high level and it also models major aspects of conventional land warfare. Because it models so many things at a high level of aggregation, THUNDER is considered a low-resolution model. Since THUNDER is designed so that the data is not built into the model, a THUNDER baseline input database is created for a specific theater combat scenario. Model data is then stored in a database made up of many data files.

3.4.4. THUNDER's Air-to-Air Submodel

THUNDER contains its own air-to-air combat submodel that models air-to-air engagements at a very aggregated level. THUNDER can be run using a high- or low-resolution setting for air-to-air combat, which determines which part of its air-to-air engagement submodel it uses to adjudicate air combat. THUNDER's low-resolution air-to-air setting allows only one aggregate PK to be set for all attacking aircraft/weapon versus target aircraft combinations for both Blue and Red. This limitation of the low-resolution setting is unrealistic and directly conflicts with the goal of the calibration process. This is because the calibration process attempts to assign the correct PK values to key THUNDER air-to-air interactions, which often involve many different aircraft from both the Blue and Red sides. If the low-resolution setting is used, there is no need to perform the calibration. Therefore, THUNDER's high-resolution setting is required for the air-to-air calibration process.

It should be noted, however, that "high-resolution" in the context of THUNDER's air-to-air model is a relative term. THUNDER's high-resolution air-to-air setting does not allow THUNDER air-to-air engagements to be adjudicated at a level of detail equal to that in BRAWLER. As stated in Section 1.3, THUNDER models many aspects of warfare, so in order to avoid being prohibitively large, its air-to-air model is less detailed than BRAWLER's.

THUNDER's high-resolution air-to-air combat process is controlled by four primary inputs that are listed and defined below.

1. Air-to-Air PK -- The probability that a particular attacking aircraft with a particular weapon will kill a specific target aircraft given that a weapon is fired during an encounter between the two. This can be written as

   $$PK(Aircraft{+}Weapon\ v\ Target) = P(Killing\ Target | Firing\ at\ Target) \qquad (3.1)$$

2. Missile Launches Per Engagement -- The number of missile launches per engagement for a specific aircraft weapon load for each engaged aircraft.

3. Relative Range Advantage -- A weapon range advantage comparison between opposing aircraft/weapon combinations given an encounter between the two has taken place.

4. Detection Capability -- The probability that an aircraft will complete an air-to-air engagement against an opponent given the opponent has been detected (ASC/XRA, 1996).

ASC/XRA has determined that inputs 1 and 2 above may be calibrated using BRAWLER. As previously mentioned, we are focusing on input 1, air-to-air PK.

In THUNDER, an attacking aircraft with a specific weapon is called a *Killer* and a targeted aircraft (regardless of weapon type) is called a *Target*. It is important to note that

THUNDER pairs an attacking aircraft with its weapon in this manner. A PK value is assigned to each Killer versus each Target. THUNDER's high-resolution air-to-air submodel adjudicates air combat using three steps, as follows:

1. A single shot probability of kill (SSPK) is calculated for each Killer versus Target combination. SSPK is defined for a Killer versus Target combination as follows:

   $$SSPK(Killer\ v\ Target) = PL(Killer) \cdot PK(Weapon\ v\ Target) \qquad (3.2)$$

   where *PL(Killer)* is the probability of a weapon launch by the killer and *PK(Weapon v Target)* is the probability of kill for the killer's weapon versus the target aircraft.

2. Next, single aircraft versus aircraft SSPK values are aggregated into attrition rates for flight versus flight and flight group versus flight group levels.

3. Finally, engagement losses are assessed by random draws from binomial distributions.

3.4.5. Calibration of THUNDER Using BRAWLER

The process selected for calibrating THUNDER using BRAWLER is the result of several iterations of evaluating THUNDER and BRAWLER results. ASC/XRA ran several experiments using THUNDER and identified four primary inputs that determine the outcome of THUNDER air-to-air engagements. They determined that it is possible to correlate two of these four inputs with BRAWLER results, as shown in Figure 3.3. This resulted in development of the methodology for the calibration of THUNDER with BRAWLER. (ASC/XRA, 1996). The process focuses on the outcome of THUNDER air-to-air engagements, not whether or not an engagement occurs (Taranto, 1995). Using the current THUNDER calibration process, a BRAWLER AKPF MOE is used as a THUNDER air-to-air PK input value for a Killer versus Target pair. When the Red and

Blue aircraft are the same in each model and the THUNDER Killer fires the same missile as its BRAWLER counterpart, then this relationship can be written as follows:

$$\textit{BRAWLER E(Missile AKPF)} = \textit{THUNDER PK (Killer v Target)} \qquad (3.3)$$

The AKPF MOE is calculated in a BRAWLER post-processing routine in a procedure explained in Section 4.4.1. The BRAWLER AKPF MOE is the average effectiveness of a missile including the effects of missile kinematics, tactics, and target maneuvering (ASC/XRA, 1996).

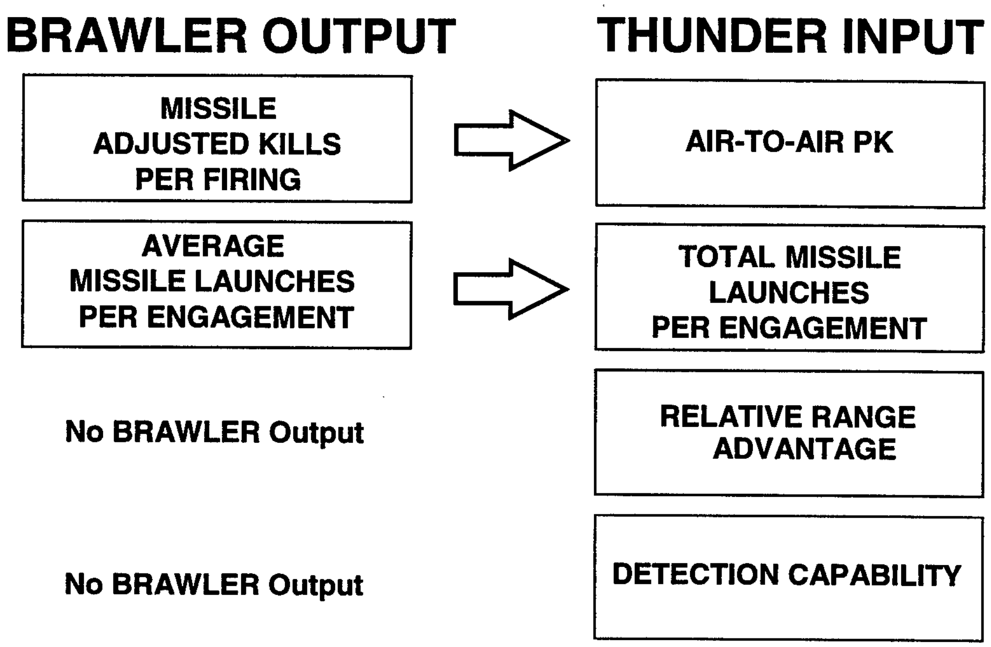


Figure 3.3. BRAWLER Output / THUNDER Input Linkages (Taranto, 1995)

3.4.6. Analysis of Calibration Options

Two calibration options were identified by ASC/XRA in order to calibrate highly aggregated (low-resolution) THUNDER air-to-air engagements with high-resolution BRAWLER engagement analysis, as follows:

1. Calibrate attrition for each distinct engagement in THUNDER, or
2. Calibrate by individual attacker/weapon performance versus an opponent.

Option 1 allows a direct correlation between BRAWLER and THUNDER engagements; however, the large number of engagement types generated by THUNDER during a multi-day campaign makes running BRAWLER for all of these possible engagements impractical, perhaps even infeasible (Taranto, 1995). Additionally, the THUNDER database structure does not allow multiple PK's for a Killer versus Target pair. This feature would be required to execute option 1.

On the other hand, the current THUNDER database structure supports option 2. The THUNDER database is built for aircraft/weapon versus aircraft combinations, which makes it necessary to calibrate PK's by individual missile system performance rather than by engagement attrition (ASC/XRA, 1996). Option 2 is far easier to implement, and still allows BRAWLER outputs and THUNDER inputs to be connected, albeit in an aggregate manner.

Option 2 describes the current THUNDER calibration process. Taranto (1995) documents several assumptions and limitations of the BRAWLER calibration process, as follows:

1. A few BRAWLER scenarios are assumed to be representative of all THUNDER air-to-air engagements.

2. The average effectiveness of a Killer versus a Target is approximately the same regardless of whether the weapon is used as the primary munition, used with different tactics, or faces the same opponent armed with different weapons.

3. The THUNDER database structure limits the air-to-air PK inputs for an Killer versus a Target to a single value. For example, for a MIG-29 armed with an AA-10A versus an F-15E, no corrections can be made to the AA-10A effectiveness when considering the F-15E armament.

Based upon findings in discussions with the THUNDER and BRAWLER analysts in ASC/XRA, we should add another limitation to this list, as follows:

4. BRAWLER's calibration of THUNDER is only valid for the specific scenario of interest, due to the fact that the distribution of engagements may change from one scenario to another.

Taranto alludes to this when he states that "types of engagements may change with changes in a concept of operations (CONOPS)." A CONOPS may be directly related to a scenario of interest.

THUNDER has been calibrated using BRAWLER by ASC/XRA most recently using BRAWLER Version 6.15. Individual air-to-air engagements that occurred in THUNDER were compared to BRAWLER MOE's and MOO's and examined from two different perspectives. From a THUNDER input data perspective, the THUNDER PK input was compared to BRAWLER AKPF output to ensure they were similar; if not, an appropriate investigation was performed and subsequent corrections were made. From an outcome perspective, they compared the outcomes from specific THUNDER engagements to BRAWLER engagements with similar force structures to ensure that they were also similar. In this manner, the air-to-air calibration process ensures that THUNDER air-to-air inputs and outputs are realistic. A simplified example of this process that focuses only on the THUNDER PK input is illustrated in Figure 3.4.

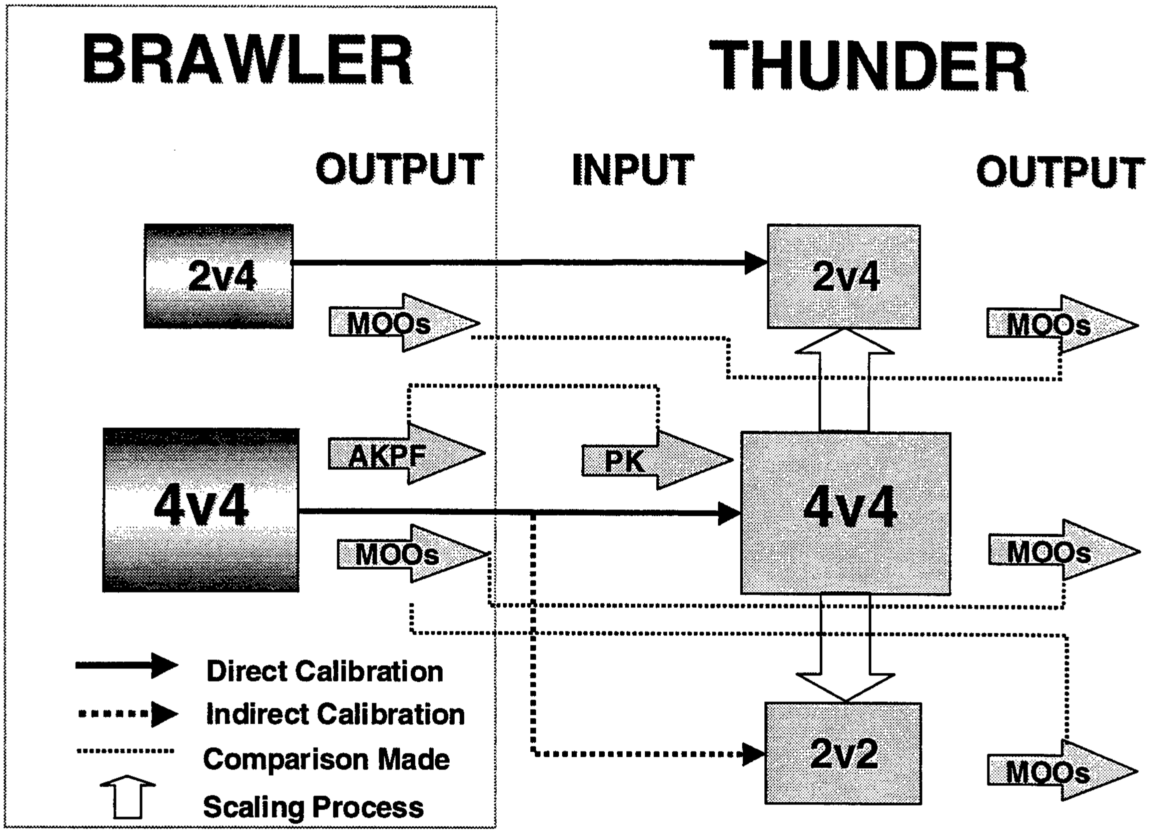


Figure 3.4. Calibration of THUNDER Outcomes and Input

A representative air-to-air engagement force structure must be chosen for each Killer/Target combination. In other words, one force structure, for example 4v4, must be chosen as a baseline that all THUNDER air-to-air engagements between the Killer and the Target are derived from, using a scaling process. This is necessary since THUNDER's database structure only allows one PK for each Killer/Target combination, as previously mentioned in this section. This is accomplished by using the most common force structure of an engagement for a Killer/Target combination that occurs in THUNDER to represent all expected engagements involving that Killer/Target combination. The BRAWLER and THUNDER users in ASC/XRA have used this method in the past. The BRAWLER users polled the THUNDER users for the most common engagement observed in the THUNDER scenario of interest for each Killer/Target combination (Williams, 1997). It should be noted that the most common force structure of an engagement is dependent upon the minimum flight sizes that THUNDER allows for both the Killer and Target aircraft. A THUNDER user sets minimum flight sizes by aircraft in *typeac.dat* and by mission in *airplan.dat*.

Recall from Section 3.4.4 that single aircraft versus aircraft SSPK values are calculated for each killer versus target combination and then aggregated into attrition rates for flight versus flight and flight group versus flight group levels. Using calibration option 2, the THUNDER single aircraft versus aircraft SSPK values are derived from the individual Killer performance versus a Target in the appropriate representative THUNDER engagement.

In the example depicted in Figure 3.4, a 4v4 engagement is THUNDER's representative engagement. Therefore, the THUNDER PK calibration using BRAWLER AKPF is performed on THUNDER 4v4 engagements using BRAWLER 4v4 engagements. Also, 4v4 BRAWLER outcomes are compared to 4v4 THUNDER outcomes and are directly calibrated from this comparison.

4v4 BRAWLER engagements may also be compared to 2v2 THUNDER engagements since the force ratio is 1:1 for both of these engagements. Often, this type of comparison is made instead of actually running the additional engagement in BRAWLER and making a direct comparison to the similar engagement in THUNDER. This is because similar outcomes should be expected, and have indeed been observed by

ASC/XRA, for both of these engagements (Logan, 1998). A 4v4 BRAWLER engagement and a 2v2 THUNDER engagement are indirectly calibrated in this manner.

In the calibration process, engagements in addition to the representative THUNDER engagement are performed in BRAWLER, such as 2v4. The outcomes of these engagements may be directly calibrated as well. The engagements that are directly calibrated are generally the most common engagements in THUNDER.

4. Methodology

4.1. Introduction

Chapter 4 discusses the methodology of the research in this thesis. Section 4.2 and 4.3 discuss specific details about BRAWLER and THUNDER, respectively, that apply to this methodology. Section 4.4 describes the ASC/XRA THUNDER calibration process that was followed in this research to ensure that the THUNDER PK input data resulting from this research has a pedigree. Section 4.5 describes the bootstrap sampling method that was used to construct an alternative confidence interval to the standard confidence interval that estimates the expected value of AKPF. Finally, Section 4.6 describes the sequential procedures for constructing these two intervals, which we have denoted Candidate S1 and Candidate S2.

4.2. BRAWLER

4.2.1. Introduction

This section highlights specific BRAWLER items that were important to this research, beginning in Section 4.2.2 with the database employed. We also describe the output variable of interest (AKPF) in Section 4.2.3. Finally, we describe the input variable that was used in this research in Section 4.2.4.

4.2.2. Version and Database

Most BRAWLER runs are made in a classified environment when performing analyses of weapon systems. This is due to the classified nature of the weapon systems being modeled. However, for this research, an unclassified version of BRAWLER denoted *Version 6.3 – Unclassified* and an unclassified database provided by the USAF's Survivability/Vulnerability Information Analysis Center (SURVIAC) were used. The unclassified BRAWLER database includes notional aircraft, missiles, and radars for both the Blue and Red sides. BRAWLER's notional Blue aircraft is loaded with notional semi-active missiles, similar to AIM-7s, and notional infrared (IR) missiles similar to AIM-9s (Taranto, 1997).

4.2.3. Output Variable

The BRAWLER output variable of interest is the adjusted kills per firing (AKPF) MOE for BRAWLER's notional IR missile. Kills per firing (KPF) for a missile type is defined as the proportion of the total number of times the missile killed its target out of the total number of times the missile was fired. To obtain AKPF, we apply a correction factor to KPF. Formal definitions of KPF and AKPF are introduced in Section 4.4.1.

BRAWLER computes an AKPF for both the IR and semi-active missiles. We decided to focus on the AIM-9-like IR missile, since the THUNDER database used in this research contains the AIM-9 missile, but not the AIM-7. For the purpose of this research, we consider BRAWLER's IR missile to be an AIM-9. The AKPF value for the AIM-9 is used as the THUNDER PK input variable for the F-15C/AIM-9 aircraft/weapon combination, which is discussed in Section 4.4.2.

4.2.4. Input Variable

BRAWLER contains three discrete pilot skill level settings that may be used as input variables: Rookie (R), Pilot (P), and Ace (A). Buschor (1998) designed an unclassified experiment using 24 different combinations of these three pilot skill level settings for the pilots of each of four Blue aircraft in a 4v4 combat engagement. In this type of engagement, the four aircraft are called a *flight*, and the aircraft are ordered. This particular size flight is the ordered set of four aircraft, denoted (1,2,3,4). The skill level combination of pilots in a four-aircraft flight is denoted by a set of four letters, one for each aircraft in the flight. For example, the case where the pilots of aircraft 1 and 2 are Pilots, aircraft 3 is an Ace, and aircraft 4 is a Rookie is denoted PPAR. We call each different combination of input variables a *scenario.*

With Buschor's permission, we used BRAWLER output data resulting from 200 replications of each of three of his scenarios, identified in Table 4.1:

Table 4.1. BRAWLER Scenarios Used

Scenario Number	Pilot Types (1,2,3,4)
1	AAAA
2	PPPP
3	RRRR

For other BRAWLER settings used in these scenarios, see Buschor (1998). In this research we used this data for the purpose of determining when enough BRAWLER runs have been performed to obtain an accurate estimate of the mean AKPF for each scenario.

4.3. THUNDER

4.3.1. Introduction

In this section we highlight the specific THUNDER items that were important in this research. First we describe the THUNDER database that was used in Section 4.3.2. We then discuss the input variable of interest, the THUNDER PK, in Section 4.3.3.

4.3.2. Version and Database

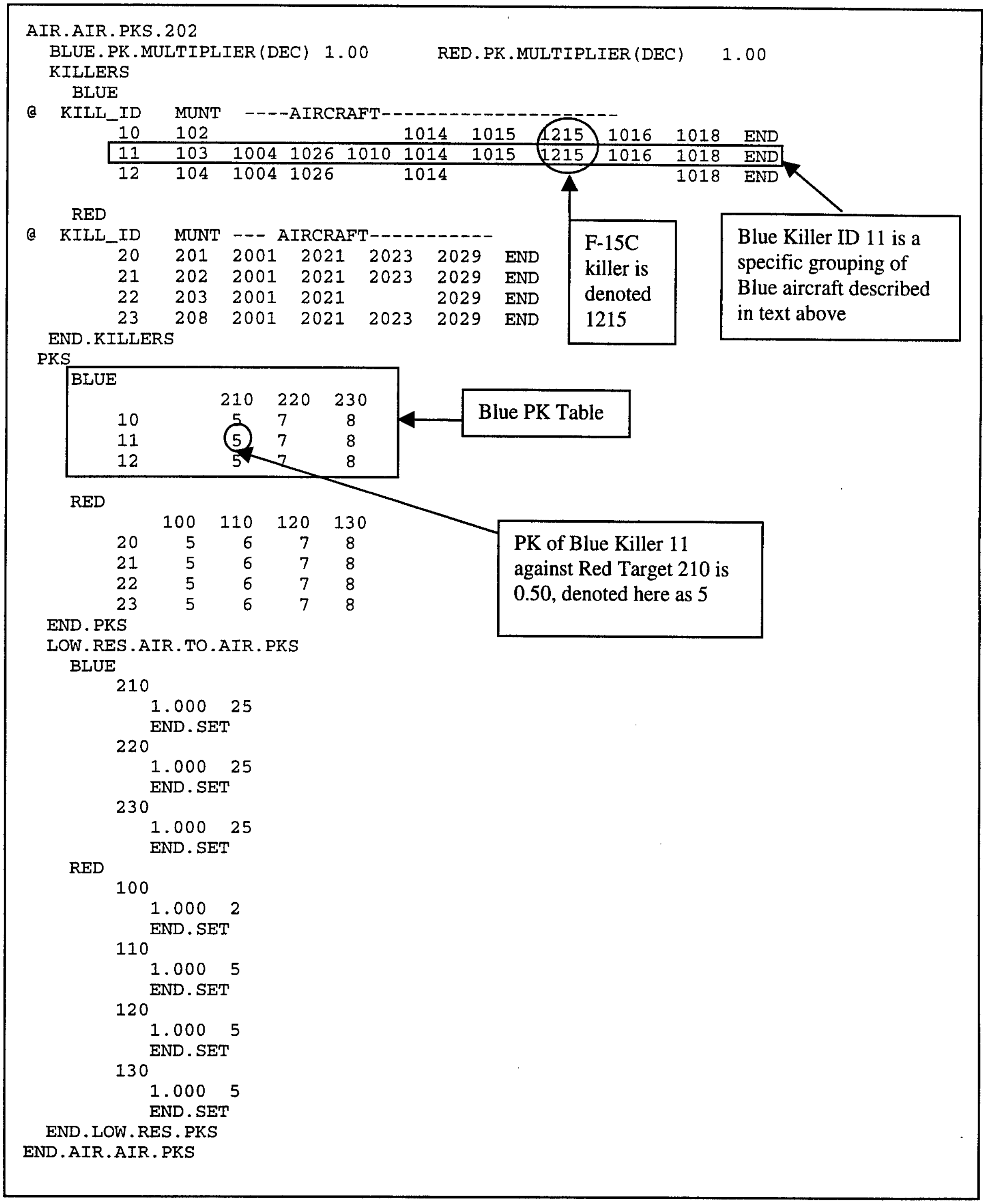
THUNDER, like BRAWLER, is run almost exclusively in a classified environment, using classified input databases. A THUNDER input database is modified as necessary to create different scenarios. During this research, we used a notional, unclassified version of a database called STORM provided by ASC/XRA. STORM simulates a Southwest Asia scenario similar to Operation Desert Storm. The version of THUNDER we used is version 6.4.2.

Because STORM is a notional database, it is much smaller and more generalized than an actual THUNDER database. For example, the types of Blue Killers are limited to the major fighter aircraft in the US inventory. Furthermore, Blue Killers are grouped together into a *Blue Killer Table* into classes depending on what weapon(s) they may fire. The same is true for the Red Killers and the *Red Killer Table*. Since the AIM-9 missile is the weapon that we are focusing on when running BRAWLER, it is appropriate to list the Blue Killers that may fire an AIM-9, which are as follows: the F-4G, EA-6B, A-10, F-14, F-15C, F-15E, F-16, and the F/A-18. All of these Blue Killers with AIM-9s are given the same killer identification number of 11 in STORM, which is called a *Killer ID*. Similarly, the Red Targets MIG-29, MIG-21, MIG-23, and F-1 are all assigned the same identification number of 210, called a *Target ID*. Similar groupings of Red Killers and Blue Targets are used.

4.3.3. Input Variable

Air-to-air PK, as defined in (3.1), is our THUNDER input variable of interest. A file named *airairpk.dat* appears in the STORM database and contains two air-to-air PK tables. One is designated the *Blue PK Table* and lists the PK value for each Blue Killer ID versus each Red Target ID. A similar table exists for Red Killers versus Blue Targets, designated the *Red PK Table*. In STORM's Blue PK Table, Blue Killer 11 is assigned a PK value of 5 (which is shorthand notation for a PK of .50) versus Red Killer 210. Since similar groupings of Blue killers/Red Targets and Red killers/Blue Targets are used, the Red PK Table looks similar to the Blue PK table. STORM's *airairpk.dat* file appears in Figure 4.1.

Each PK in the Blue and Red PK tables represents the probability of a Killer killing the Target, given a weapon firing, for all THUNDER air-to-air engagements between the Killer and the Target. THUNDER's database structure only allows one PK for each Killer/Target combination, as discussed in Section 3.4.6. Also, the PK input value for the Killer versus the Target is the same regardless of the weapon load the Target is carrying or the weapons the Target has fired.



```
AIR.AIR.PKS.202
  BLUE.PK.MULTIPLIER(DEC) 1.00      RED.PK.MULTIPLIER(DEC)   1.00
  KILLERS
    BLUE
@  KILL_ID   MUNT  ----AIRCRAFT---------------------
        10   102                 1014  1015  1215  1016  1018  END
        11   103  1004 1026 1010 1014  1015  1215  1016  1018  END
        12   104  1004 1026      1014                    1018  END

    RED
@  KILL_ID   MUNT --- AIRCRAFT-----------
        20   201  2001  2021  2023  2029  END
        21   202  2001  2021  2023  2029  END
        22   203  2001  2021        2029  END
        23   208  2001  2021  2023  2029  END
  END.KILLERS
 PKS
    BLUE
                210  220  230
        10        5    7    8
        11        5    7    8
        12        5    7    8

    RED
              100  110  120  130
        20      5    6    7    8
        21      5    6    7    8
        22      5    6    7    8
        23      5    6    7    8
  END.PKS
  LOW.RES.AIR.TO.AIR.PKS
    BLUE
        210
           1.000  25
           END.SET
        220
           1.000  25
           END.SET
        230
           1.000  25
           END.SET
    RED
        100
           1.000  2
           END.SET
        110
           1.000  5
           END.SET
        120
           1.000  5
           END.SET
        130
           1.000  5
           END.SET
  END.LOW.RES.PKS
END.AIR.AIR.PKS
```

F-15C killer is denoted 1215

Blue Killer ID 11 is a specific grouping of Blue aircraft described in text above

Blue PK Table

PK of Blue Killer 11 against Red Target 210 is 0.50, denoted here as 5

Figure 4.1. *airairpk.dat* (AFSAA, 1995)

4.4. Calibration of THUNDER PK Using BRAWLER AKPF

4.4.1. Calculating AKPF

As mentioned in Section 3.4.6, the BRAWLER output MOE of kills per firing (KPF) for an engagement is transformed into the THUNDER air-to-air PK input value. The BRAWLER KPF output is the average effectiveness of a missile, including such effects as missile kinematics, tactics, and target maneuvering (ASC/XRA, 1996). The BRAWLER KPF output is transformed into AKPF, which becomes the THUNDER PK input for each Killer versus Target pair. This process is accomplished in four steps:

1. A representative air-to-air engagement force structure (i.e. 4v4) is chosen for each Killer/Target combination.
2. A KPF MOE is obtained.
3. An AKPF MOE is obtained by applying a correction factor to KPF.
4. A THUNDER PK is assigned.

In step 1, the force structure is chosen as described in Section 3.4.6. Step 2 requires running BRAWLER a sufficient number of times and calculating the mean KPF value for those replications. The KPF MOE is calculated in a BRAWLER post-processing routine created by Taranto called *STATS*, which appears in Appendix C. *STATS* processes each of the BRAWLER log files sequentially and totals all the missile firings, fuzings, and kills over all the BRAWLER replications performed. KPF is then written to a missile effectiveness report in a file called *STATS.prt*. An example of this report appears in Appendix D. The KPF value is displayed as “Kills/Firing” in a report in *STATS.prt* titled “Missile Effectiveness Report.” The calculation of KPF is straightforward, according to:

$$KPF = \frac{\sum_{i=1}^{N} Kills}{\sum_{i=1}^{N} Firings} \tag{4.1}$$

where N is the number of BRAWLER replications performed. Since all of the equations in this chapter involve sums calculated over N replications, we drop the summation indices and simply write:

$$KPF = \frac{\sum Kills}{\sum Firings}\ . \tag{4.2}$$

As shown in (4.3), it is not necessary to first calculate KPF to obtain AKPF, but calculating KPF makes the calculation of AKPF more intuitive.

In step 3, an AKPF MOE is obtained by applying a correction factor to KPF. This correction factor in step 3 is necessary because sometimes a missile fuzes on a dead target, alive when the missile was fired but recently killed by another missile. These events are denoted $Fuzings_{DEAD}$ and are counted in the *STATS* routine. $Fuzings_{DEAD}$ are not counted as kills when calculating KPF; however they are counted as kills when calculating AKPF. The reason for this is that AKPF measures the proportion of missiles that fuzed on their targets, live or dead. Because of this, AKPF gives a better representation of the missile's overall effectiveness than KPF does, and that is what we wish to represent when we pass PK data to THUNDER. The formula for AKPF is:

$$AKPF = \frac{\sum Kills}{\sum Firings - \frac{\sum Fuzings_{DEAD} \cdot \sum Firings}{\sum Fuzings}}\ . \tag{4.3}$$

or more intuitively:

$$AKPF = KPF \cdot \frac{\sum Fuzings}{\sum Fuzings - \sum Fuzings_{DEAD}} \cdot \tag{4.4}$$

The derivation of (4.4) from (4.3) appears in Appendix H. Note that $\mathrm{AKPF} \geq \mathrm{KPF}$. The AKPF value appears in another *STATS.prt* report called "THUNDER Missile Effectiveness Report." As mentioned in Section 3.4.6, the AKPF output is only representative of the specific engagement that was run (i.e. 4v4). We are now ready to complete this process with step 4.

In step 4, the THUNDER PK is assigned the resulting BRAWLER AKPF in step 3 for the THUNDER representative engagement. For example, if the representative engagement is 4v4, an expression for the THUNDER PK is as follows:

$$PK = AKPF_{4v4} \cdot \tag{4.5}$$

4.4.2. Strategy for Experiment

Before starting the air-to-air calibration experiment, we consulted a member of ASC/XRA's BRAWLER analysis group, Mr. Larry Taranto. He suggested that in order to make our research as realistic as possible with respect to how the calibration process is actually performed that we take the following steps:

1. Investigate how many air-to-air encounters the F-15C force typically makes in the STORM scenario in THUNDER to ensure that there are a highly significant number of encounters. If not, we should consider choosing the Blue aircraft that had the most air-to-air encounters. This aircraft is the best choice of a Blue aircraft to calibrate first. This is because the more encounters an aircraft has determines, in part, its significance in the campaign. The more significant an aircraft is in the campaign, the more necessary it is to calibrate this aircraft's THUNDER engagements with BRAWLER engagements.

2. Determine which Red aircraft the F-15C (if chosen first) most frequently engages. This aircraft is the best choice of a Red aircraft to calibrate first, for the same reason as in step 1 above.

3. Let BRAWLER's notional Blue aircraft represent the F-15C and its Red aircraft represent the F-15C 's most frequently engaged Red aircraft. Then data from BRAWLER using the notional weapon systems can be used to populate the notional F-15C database in THUNDER (Taranto, 1997).

Taking Mr. Taranto's advice, we performed 30 THUNDER replications of a 30-day war using the STORM database. We determined that the F-15C has the most air-to-air encounters of any Blue aircraft, and it engages the MIG-29 Red aircraft most frequently. Therefore, for the purpose of this research, we consider BRAWLER's Blue aircraft an F-15C and its Red aircraft a MIG-29. Then BRAWLER AKPF data is used to populate the F-15C/AIM-9 aircraft/weapon combination (Killer) entry in the THUNDER PK table.

4.5. The Bootstrap Sampling Method

4.5.1. Introduction

Efron (1979) popularized the idea of the bootstrap as a way to estimate a sampling distribution. The bootstrap is a technique that uses computer processing power to estimate the accuracy of statistical estimates by simplifying certain complex calculations of traditional statistical theory (Efron and Tibshirani, 1993). The bootstrap method involves analyzing pseudo-data obtained by repeatedly and randomly *resampling* the original data. In Section 4.5.2 we describe the bootstrap sampling process and in Section 4.5.3 we describe methods for constructing bootstrap confidence intervals.

4.5.2. Bootstrap Sampling

Much of statistical inference is describing the relationship between a sample and the population from which the sample was drawn (Hall, 1992). The bootstrap method, like other statistical inference methods, requires a random sample. A *sample* $X = (x_1, x_2, \ldots, x_n)$ is an unordered collection of *n* data points randomly drawn from a population that can be represented by a vector X that represents the sample. The x_i's (the elements of the vector X) are independent and identically distributed (iid) random variables with some population distribution function. This population distribution function of X is denoted *F*. A sample that is obtained by randomly resampling with replacement from the original sample is called a *bootstrap sample*. Formally defined, the j^{th} bootstrap sample $X_j^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an unordered collection of *n* members of X that are drawn randomly from X with replacement, where the star (*) notation indicates pseudo-data rather than actual data. The fact that we are sampling with replacement means that X_j^* may (and, in fact, almost certainly *will*) contain one or more duplicates of the original data points in X. The bootstrap method can be illustrated with the following simple example.

Suppose our original sample $X =$ *(0,10,0,40,100,33,33,0,100,50)* represents $\text{AKPF} \cdot 100\%$ outputs from ten BRAWLER runs. If we take ten bootstrap samples from X, we might obtain the following results shown in Table 4.2:

Table 4.2. Ten Bootstrap Samples

j	x_1^*	x_2^*	x_3^*	x_4^*	x_5^*	x_6^*	x_7^*	x_8^*	x_9^*	x_{10}^*
1	0	33	100	100	40	100	40	33	40	50
2	40	10	0	33	10	100	50	0	0	100
3	33	33	100	0	33	50	40	0	0	10
4	40	10	40	33	0	33	0	10	0	0
5	33	10	50	100	33	50	100	50	50	33
6	40	33	33	50	10	100	0	50	100	50
7	33	50	0	0	100	40	0	100	33	100
8	100	33	50	100	0	0	50	33	33	33
9	33	0	100	0	0	10	100	33	33	10
10	33	33	100	50	0	33	40	100	100	0

When we form bootstrap samples in this manner, it should be noted that we are treating the original sample X as if it represents a population. In fact, one interpretation of the main idea behind the bootstrap is to "let the sample play the role of the population" (Kim, 1996). Another interpretation is that when we use the bootstrap method, "it is as if we are sampling from an infinite population with a composition that exactly matches that of the sample that we drew" (Friedman & Friedman, 1995). So, when we use the bootstrap method, we treat X as if it were the population from which X^* was drawn. While it may be possible to compute the resampling distribution of X analytically, it is often quicker and easier to simply "build up enough of a picture" of the distribution of X by using a simple computer simulation (Ripley, 1987).

In a similar manner as the population distribution function was denoted *F*, the distribution function of the sample X^* is denoted $\hat{F}$. $\hat{F}$ is called the *empirical distribution function*, which is conditional on *F*. We use the hat (^) notation to remind us that $\hat{F}$ gives us an estimate of *F*.

The bootstrap method is often used to estimate a parameter, such as the mean, of a population with a specific distribution. A parameter θ of a population is a functional of the distribution function *F*, which can be denoted $\theta = \theta(F)$. When we treat our sample like a population, an estimated parameter of the distribution function of the sample should be a good estimator of the parameter of the population. We obtain this estimated parameter as follows:

1. Form a bootstrap sample by randomly sampling *n* times, with replacement, from the original sample of size *n*.

2. Repeat this process *B* times, where *B* is a large number, so that we obtain *B* bootstrap samples of size *n*.

3. For j = 1 to *B*, calculate a parameter $\hat{\theta}_j$ of the j^{th} bootstrap sample that is based on the bootstrap distribution of $\hat{\theta} = \hat{\theta}(F)$. These parameters $\hat{\theta}_j$ give us a good idea of what the distribution of θ looks like.

It should be noted that the bootstrap method is a stochastic process and if the procedure is repeated, slightly different results should be expected. Let us return to our example and suppose that we are interested in estimating θ, where θ is the population mean. Then we would be interested in finding $\hat{\theta}_j$, the mean of bootstrap sample j. These values have been added to our original table (Table 4.2) and appear in Table 4.3.

Table 4.3. Ten Bootstrap Samples and Their Respective Means

j	x_1^*	x_2^*	x_3^*	x_4^*	x_5^*	x_6^*	x_7^*	x_8^*	x_9^*	x_{10}^*	$\hat{\theta}_j$
1	0	33	100	100	40	100	40	33	40	50	53.6
2	40	10	0	33	10	100	50	0	0	100	34.3
3	33	33	100	0	33	50	40	0	0	10	29.9
4	40	10	40	33	0	33	0	10	0	0	16.6
5	33	10	50	100	33	50	100	50	50	33	50.9
6	40	33	33	50	10	100	0	50	100	50	46.6
7	33	50	0	0	100	40	0	100	33	100	45.6
8	100	33	50	100	0	0	50	33	33	33	43.2
9	33	0	100	0	0	10	100	33	33	10	31.9
10	33	33	100	50	0	33	40	100	100	0	48.9

We put our example aside for now, but will return to it in Section 4.5.3.4

The reader may have reservations about this process, as we did when first faced with the idea of the bootstrap. In fact, Ripley (1987) goes so far as to write, "At first sight this is a preposterous idea, but it has been shown to work well in a wide variety of problems." The key assumption in the bootstrap method is that the sample represents the population. This heavy leverage on the original sample is a shortcoming of the bootstrap method (Friedman and Friedman, 1995). The advantage of using bootstrap sampling is that we can obtain an estimate of the parameter of a distribution of interest without having to make any assumptions about its distribution.

4.5.3. Bootstrap Confidence Intervals

4.5.3.1. Introduction

Many statistical inference procedures involve estimating a parameter(s) θ of a distribution. An interval estimator, more commonly known as an approximate *confidence* *interval*, is a procedure that specifies a method for using a sample to form a specific interval. This interval is designed such that, ideally, it possesses two attributes:

1. The interval contains the parameter θ , and
2. The interval is relatively narrow.

The endpoints of the interval are called the *upper* and *lower confidence limits*, and since they are functions of a random sample, they themselves are random quantities. Therefore, we cannot be certain that one interval, which is calculated from one sample, contains θ . However, if we use a proper interval estimator, we can be confident that, when we repeatedly collect samples, the interval generated from each sample will contain θ with probability *C*. A *confidence coefficient C* is defined as the approximate fraction of the time that, in repeated sampling, these intervals contain θ ; or in other words, the approximate probability that the confidence interval will contain θ . Thus, if *C* for our interval is high (close to 1), then we can be highly confident that a single confidence interval calculated from a single sample contains θ (Wackerly, Mendenhall, and Scheaffer, 1996).

4.5.3.2. Standard Confidence Intervals

The favorite approximate confidence interval, by far, is the *standard confidence interval* (DiCiccio and Efron, 1996). Suppose we have one random sample *X* from a population with an unknown distribution *F*. Furthermore, let $\hat{\theta} = \theta(\hat{F})$ be the estimate of a parameter $\theta = \theta(F)$, and let $\hat{\sigma}_{\hat{\theta}}$ be the *standard error* of $\hat{\theta}$. The standard error is a useful measure of the accuracy of $\hat{\theta}$ as an estimate of θ . Standard errors are commonly used to develop approximate confidence intervals for θ . An expression for $\hat{\sigma}_{\hat{\theta}}$ is developed in the equations that follow.

The standard deviation $\hat{\sigma}$ of our sample X of x_i's is defined as

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.6}$$

where $\bar{x}$ is the mean of the sample and n is the sample size. We continue to use the hat notation to indicate that $\hat{\sigma}$ is an estimate of the population's standard deviation σ . From this point forward, we shall assume that the parameter θ that we are trying to estimate is the mean, so in this case, $\hat{\theta} = \bar{x}$. Then we can replace (4.6) with the following:

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\theta})^2} \tag{4.7}$$

The standard error of $\hat{\theta}$ is equivalent to the standard deviation of $\hat{\theta}$, which we have previously denoted $\hat{\sigma}_{\hat{\theta}}$. An expression for $\hat{\sigma}_{\hat{\theta}}$ is

$$\hat{\sigma}_{\hat{\theta}} = \frac{\hat{\sigma}}{\sqrt{n}} \tag{4.8}$$

The so-called *large-sample result* maintains that under most circumstances, as the sample size n grows large, the distribution of $\hat{\theta}$ approaches normality, with mean near θ and variance near $\hat{\sigma}^2$, written $\hat{\theta} \sim N(\theta, \hat{\sigma}^2)$. From this result we define Z as the following:

$$Z = \frac{\hat{\theta} - \theta}{\hat{\sigma}} \tag{4.9}$$

where $Z \sim N(0,1)$ as the sample size $n \to \infty$. We say that Z is distributed according to the *standard normal* distribution or *z* distribution. The standard confidence interval is derived from the assumption that $Z \sim N(0,1)$. However, this assumption is only valid for an infinite sample size. For $\hat{\theta} = \overline{x}$, which we have already assumed to be the case, we use a better approximation derived by Gosset (1908):

$$Z = \frac{\hat{\theta} - \theta}{\hat{\sigma}} \sim t_{(n-1),} \tag{4.10}$$

which is distributed according to the *Student's t-distribution* with *n-1* degrees of freedom. We may now define a $100 \cdot (1-\alpha)\%$ standard confidence interval as follows:

$$\hat{\theta} \pm t_{(n-1,a/2)} \cdot \hat{\sigma}_{\hat{\theta}} \tag{4.11}$$

or equivalently as

$$\left(\hat{\theta} - t_{(n-1,a/2)} \cdot \hat{\sigma}_{\hat{\theta}}, \hat{\theta} + t_{(n-1,a/2)} \cdot \hat{\sigma}_{\hat{\theta}}\right) \tag{4.12}$$

where $\alpha = 1 - C$ is called the level of significance and $t_{(n-1,a/2)}$ is the $100 \cdot \alpha/2$ percentile of a *t* distribution with n-1 degrees of freedom. We call the distance from $\hat{\theta}$ to each endpoint of the interval the half-length of the confidence interval and denote it as

$$hl(n,\alpha) = t_{(n-1,\alpha/2)} \cdot \hat{\sigma}_{\hat{\theta}} \tag{4.13}$$

where *hl* is a function of *n*, the sample size, and α, the significance level.

To help transition towards how (and why) the bootstrap method is used to generate confidence intervals, let us consider representing the $100 \cdot (1-\alpha)\%$ interval as follows:

$$\left(\hat{\theta}^{(\alpha/2)}, \hat{\theta}^{(1-\alpha/2)}\right) \tag{4.14}$$

where $\hat{\theta}^{(\alpha/2)}$ and $\hat{\theta}^{(1-\alpha/2)}$ are the $100 \cdot \alpha/2$ and $100 \cdot (1-\alpha/2)$ percentiles, respectively, of the distribution of $\hat{\theta}$. Now we may define the length of a $100 \cdot (1-\alpha)\%$ confidence interval as

$$length = \hat{\theta}[1-\alpha/2] - \hat{\theta}[\alpha/2] \tag{4.15}$$

We may also define the shape of the interval in terms of asymmetry around $\hat{\theta}$, or alternatively, skewness, as

$$shape = \frac{\hat{\theta}[1-\alpha/2] - \hat{\theta}}{\hat{\theta} - \hat{\theta}[\alpha/2]} \tag{4.16}$$

Therefore, an interval that is more dense to the left of $\hat{\theta}$ (skewed right) has a shape greater than 1, while the opposite case (skewed left) has a shape less than 1. A symmetric interval has a shape equal to 1. We have already seen from (4.11) that the standard confidence interval is symmetric.

Standard confidence intervals always have shape equal to 1, and it is in this way that they can be quite inaccurate in practice (DiCiccio and Efron, 1996). Using a standard confidence interval requires that we make assumptions based on a normal distribution, which may not be appropriate. We show in Sections 4.5.3.3 and 4.5.3.4 that this symmetric property is not necessarily the case with bootstrap confidence intervals. This characteristic is a potential asset of the bootstrap method.

4.5.3.3. Bootstrap-*t* Confidence Intervals

The bootstrap sampling method may be used to construct an approximate confidence interval for an estimated parameter $\hat{\theta}$. This parameter is estimated by a statistic that is a function of *n* independent and identically distributed (iid) observations. The process to construct a bootstrap-*t* confidence interval begins as follows:

1. Generate *B* independent bootstrap samples of size *n* using the empirical distribution of the original *n* observations.
2. Obtain the empirical distribution of the statistic from these B independent bootstrap samples.

For sufficiently large *B* one obtains an arbitrarily good approximation to a distribution that is a good estimate of the true distribution of the statistic.

By using the bootstrap procedure we can obtain accurate intervals without having to make assumptions about a standard normal distribution as in (4.10). The idea of the bootstrap-*t* interval is to estimate the percentiles of *Z* by bootstrapping. The bootstrap-*t* interval procedure estimates the distribution of *Z* by bootstrap sampling from the data. By doing this, the bootstrap-*t* interval automatically adjusts for skewness in the underlying population. We compute a bootstrap value of *Z* for each bootstrap sample *b* using the following equation:

$$Z^*(b) = \frac{\hat{\theta}^*(b) - \hat{\theta}^*}{\hat{\sigma}_{\hat{\theta}}{}^*(b)} \tag{4.17}$$

where $\hat{\theta}^*(b)$ is the estimate of $\hat{\theta}$ for the bootstrap sample x^{*b}, $\hat{\sigma}^*_{\hat{\theta}}$ is the estimated standard error of $\hat{\theta}^*$ for the bootstrap sample x^{*b} and $\hat{\theta}^* = (1/B) \cdot \sum_{b=1}^{B} \hat{\theta}^*(b)$ (DiCiccio and Efron, 1996). The $(\alpha/2 \cdot 100)$ percentile (%) of $Z^*(b)$ is estimated by the value $t_{\alpha/2}$ such that

$$Z^*(b) \leq t_{\alpha/2} = \alpha/2 \cdot B \tag{4.18}$$

where α is the significance level.

The estimate of the $(\alpha/2 \cdot 100)\%$ point of Z is the $(\alpha/2) \cdot B$ ordered value of $Z^*(b)$ and the estimate of the $[(1-\alpha/2) \cdot 100)]\%$ point is the $(1-\alpha/2) \cdot B$ ordered value of $Z^*(b)$. The bootstrap-*t* confidence interval then is

$$[(\hat{\theta} - t_{(1-\alpha/2)} \cdot \hat{\sigma}_{\hat{\theta}}), (\hat{\theta} - t_{(\alpha/2)} \cdot \hat{\sigma}_{\hat{\theta}})] \tag{4.19}$$

4.5.3.4. Bootstrap Percentile Intervals

The bootstrap percentile interval method is another simple method for assigning an approximate confidence interval to a real-valued parameter $\theta = \theta(F)$ that is based upon the bootstrap distribution of $\hat{\theta} = \hat{\theta}(F)$. In section 4.5.3.2 we considered representing the $100 \cdot (1-\alpha)\%$ confidence interval in the form of Equation (4.15). Let $\hat{F}(t) = P\{\hat{\theta}^* \leq t\}$ be the empirical distribution function of the bootstrap distribution of $\hat{\theta}^*$. We then define $\theta^{(\alpha/2)} = \hat{F}^{-1}(\alpha/2)$ and $\theta^{(1-\alpha/2)} = \hat{F}^{-1}(1-\alpha/2)$. The percentile method allows us to take

$$\left(\theta^{(\alpha/2)}, \theta^{(1-\alpha/2)}\right) \tag{4.20}$$

as an approximate $100 \cdot (1-\alpha)\%$ central confidence interval for θ (Efron, 1982). This is illustrated in Figure 4.2.

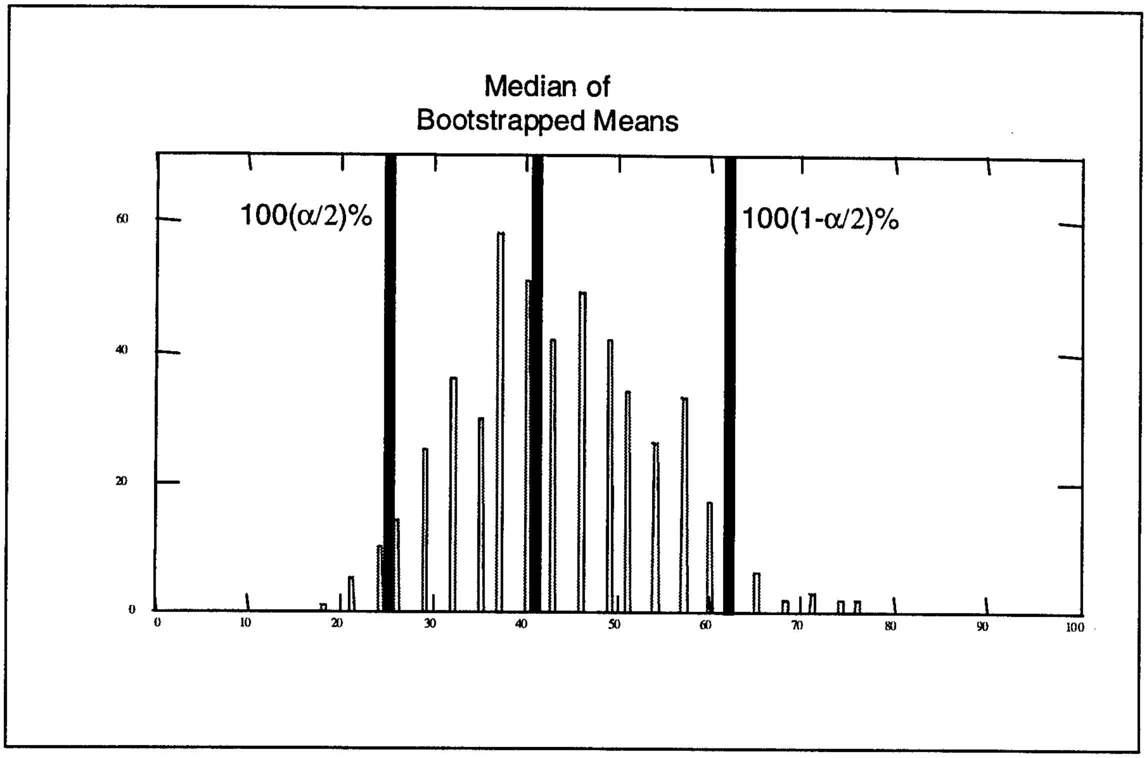


Figure 4.2. Bootstrap Percentile Interval

If B = *501* and $\alpha = .10$, which is the case in the program *enoughruns*, then $\theta^{(\alpha/2)}$ is the 26th ordered value of the replications and $\theta^{(1-\alpha/2)}$ is the 476th. We discuss this further in Section 4.6.3.

Let us return to our example and suppose that we wish to find an 80% confidence interval for θ, the population mean. We start by sorting our 10 samples by the value of $\hat{\theta}_j$, the mean of bootstrap sample *j*, as shown in Table 4.4.

Table 4.4. Ten Bootstrap Samples Sorted by Mean

j	x_1^*	x_2^*	x_3^*	x_4^*	x_5^*	x_6^*	x_7^*	x_8^*	x_9^*	x_{10}^*	$\hat{\theta}_j$
4	40	10	40	33	0	33	0	10	0	0	16.6
3	33	33	100	0	33	50	40	0	0	10	29.9
9	33	0	100	0	0	10	100	33	33	10	31.9
2	40	10	0	33	10	100	50	0	0	100	34.3
8	100	33	50	100	0	0	50	33	33	33	43.2
7	33	50	0	0	100	40	0	100	33	100	45.6
6	40	33	33	50	10	100	0	50	100	50	46.6
10	33	33	100	50	0	33	40	100	100	0	48.9
5	33	10	50	100	33	50	100	50	50	33	50.9
1	0	33	100	100	40	100	40	33	40	50	53.6

A 80% bootstrap percentile interval for $\hat{\theta}$ gives us an approximate 80% confidence interval for θ. We can construct a bootstrap 80% percentile interval for $\hat{\theta}$ using the 10th percentile and the 90th percentile of the $\hat{\theta}_j$'s, which are 16.6 and 50.9, respectively. An 80% bootstrap percentile interval for $\hat{\theta}$ based on our 10 bootstrap samples is (16.6,50.9). However, in practice, we would never use such a small number of bootstrap samples, since it is desirable (and very easy) to obtain a large number B of bootstrap samples. In comparison, a standard 80% confidence interval for $\hat{\theta}$ based on our original sample of size 10 according to (4.11) is $36.6 \pm 16.6 = (20.0, 53.2)$.

4.6. Sequential Procedures for Determining Run Length

4.6.1. Introduction

In this section we discuss how sequential procedures for determining simulation run length are applied to the problem at hand: obtaining a sufficiently precise estimate of

AKPF to use to calibrate a THUNDER PK. First, Section 4.6.2 introduces a sequential procedure presented in Law and Kelton (1991) that makes use of a standard confidence interval. Section 4.6.3 introduces a sequential procedure involving bootstrapping that is used in Johnson and Lucas' (1994) program *enoughruns* to determine when enough BRAWLER runs have been performed to obtain an accurate estimate of exchange ratio. This procedure is a predecessor to the procedure used in Candidate S2. Section 4.6.4 formally introduces Candidate S2, which is based on the procedure in *enoughruns*. Finally, Section 4.6.5 discusses how to prepare BRAWLER data for use with Candidate S2.

4.6.2. Candidate S1

Law and Kelton (1991) present a sequential procedure for obtaining an estimate of the mean θ with a specified relative error γ and a confidence level α that allows the simulation user to only perform as many replications as necessary. We choose an initial number of replications $n_0 \geq 2$ and perform the following procedure:

1. Perform n_0 replications of the simulation and set. $n = n_0$.
2. From the sample of outputs $(x_1, x_2, \ldots, x_n)$, compute the sample mean $\bar{x}(n)$ and the half-length of the confidence interval $hl(n,\alpha)$.
3. Determine if the following inequality holds:

$$\frac{hl(n,\alpha)}{|\bar{x}(n)|} \leq \gamma', \tag{4.21}$$

where ε is the observed relative error and $\gamma' = \gamma/(\gamma+1)$ is the adjusted relative error needed to obtain an actual relative error of γ. If so, stop performing replications and use

$\bar{x}(n)$ as an estimate for θ. If not, set $n = n+1$, perform another replication, and return to step 2.

Law and Kelton recommend using this sequential procedure with $n_0 \geq 10$ and $\gamma \leq 0.15$. After testing this procedure on many different simulation models, they found that if these recommendations are followed, the estimated proportion of time that θ was covered using a 90% confidence interval was never less than 0.864.

4.6.3. Candidate S2 Predecessor

Johnson and Lucas' program *enoughruns* is the predecessor to the procedure that we will use to construct a bootstrap percentile interval for the AKPF MOE. The program focuses specifically on exchange ratios and automatically determines if enough BRAWLER runs have been completed, based on the stopping criterion defined later in this section. The program *enoughruns* is automatically executed between each BRAWLER run after 20 runs have been performed.

Recall that the equation for calculating the exchange ratio for a replication *i* (ER_i) is given by (2.1). To use bootstrap sampling, we must be able to form bootstrap samples from an original sample of data from individual runs. The ER_i's from each replication *i* are such a sample. It might seem that (2.1) would allow us to bootstrap sample ER_i's. However, referring to (2.1), it should be clear that ER_i has a lower bound zero and no upper bound, written $ER \in [0,\infty)$, since as $BK_i \to 0$, $ER_i \to \infty$.

In the BRAWLER scenarios used in this research, as well as many other BRAWLER scenarios, one or more replications result in an unbounded ER_i. Since, in general, we can not form a sample of *N* bounded ER_i's, we should not attempt to bootstrap a sample of ER_i's.

We observe that ER_i is simply an aggregate value that represents the two values RK_i and BK_i. We form a sample of *N* ordered pairs (RK_i, BK_i) where $i \in [1, N]$ and denote this sample (RK, BK). We bootstrap from this sample, forming *B* bootstrap samples. The *b*th bootstrap sample $(RK^*(b), BK^*(b))$contains the ordered pairs $(RK_i^*(b), BK_i^*(b))$ where $b \in [1, B]$ and $i \in [1, N]$. We then obtain the mean exchange ratio $ER^*(b)$ for each of the *b* bootstrap samples by (4.22), which is based on (2.2).

$$ER^*(b) = \frac{\sum_{i=1}^{N} RK_i^*(b)}{\sum_{i=1}^{N} BK_i^*(b)} \tag{4.22}$$

This is the technique that Johnson and Lucas use in *enoughruns.* In their program, the following steps occur:

1. $B = 501$ bootstrap samples $(RK^*(b), BK^*(b))$ of size $N \times 2$ are formed from the original sample of *N* ordered pairs (RK_i, BK_i).

2. The mean exchange ratio $ER^*(b)$ for each of the 501 bootstrap samples is calculated using (4.22).

3. The 501 $ER^*(b)$'s are sorted in ascending order.

4. The following statistics are calculated for the 501 $ER^*(b)$'s.

   c. The median, or 50th percentile, $ER^*(b)^{(.50)}$, which is the 251st ordered value of the sorted $ER^*(b)$'s,

   d. The 5th percentile, $ER^*(b)^{(.05)}$, which is the 26th ordered value, and

e. The 95th percentile, $ER^*(b)^{(.95)}$, which is the 476th ordered value.

5. Stop performing runs if both $ER^*(b)^{(.05)}$ and $ER^*(b)^{(.95)}$ are within $\pm 10\%$ of $ER^*(b)^{(.50)}$. Otherwise continue.

In other words, this stopping criterion is met when a 90% bootstrap percentile interval with a relative error less than .10 is obtained.

The procedure is completely automated. If the stopping criterion is met, *enoughruns* returns a value of "yes," indicating that enough runs have been performed and BRAWLER replications are terminated. If the stopping criterion is not met, *enoughruns* returns a value of "no," and another BRAWLER replication is launched. Since the bootstrap method is a stochastic process, we may infer that if this procedure were repeated on the same set of BRAWLER data, a different result ("yes" versus "no") may be obtained on the same run. The procedure that Johnson and Lucas use in *enoughruns* is illustrated in Figure 4.3.

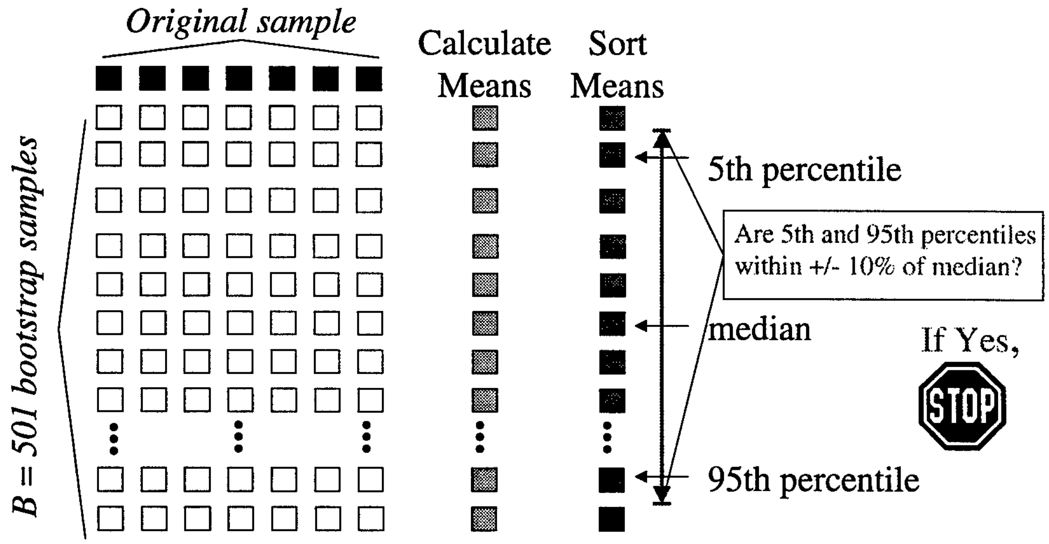


Figure 4.3. Procedure Used in *enoughruns*

4.6.4. Candidate S2

The procedure applied in this research is based on Johnson and Lucas' procedure in *enoughruns* that was discussed in Section 4.6.3; therefore, the two procedures are very similar. However, there is a key difference in the way we form a sample to bootstrap from. This is because we must consider not only the AKPF of each run, but also the number of firings in each run (Firings). This can be illustrated with a simple example. Suppose we have three BRAWLER AKPF MOE's from three different BRAWLER outcomes, and the MOO's of interest are the number of kills (Kills) and Firings. This data appears in Table 4.5:

Table 4.5. MOEs and MOOs from Three Different BRAWLER Outcomes

MOE	MOO	
AKPF	Kills	Firings
0	0	1
0	0	10
0	0	0

Although all of these outcomes result in an AKPF MOE of zero, they are clearly different. If we only consider the AKPF values from each run, we might attempt to bootstrap AKPF's from a sample of size $N = 3$ (the number of BRAWLER outcomes or runs). If we do this, we give each of these three AKPF's equal weight, which they clearly should not have. Rather, an AKPF based on 10 firings should carry 10 times the weight of an AKPF based on 1 firing, and an AKPF based on zero firings should carry no weight.

We need to collect two values for each run, AKPF and Firings, and store them in an $N \times 2$ matrix. We then create one vector of AKPF's that accounts for the number of firings associated with each AKPF. This is a key difference from Johnson and Lucas' procedure. We use our $N \times 2$ matrix to create an equal weight (with respect to Firings) vector of AKPF's of size $\sum_{i=1}^{N} Firings_i$. In this vector, which is denoted the AKPF vector, the AKPF for each replication is repeated a number of times equal to the number of Firings in that replication, which allows us to preserve information about the number of missile firings in each replication. We now treat the new AKPF vector as our original sample and bootstrap from this vector. For example, AKPF output from the first five replications of a notional BRAWLER scenario, denoted Scenario X, appears in Table 4.6. For illustrative purposes we have also displayed the Kills in each run.

Table 4.6. Kills, Firings and AKPF Matrix

Replication	Scenario X		
Number	Kills	Firings	AKPF
1	1	2	0.50
2	0	0	0.00
3	1	2	1.00
4	1	4	0.25
5	0	2	0.00

Note that in replication number 3, although there were 2 firings but only 1 kill, the AKPF is 1.00. This is due to the fact that in this replication, one of the missiles fired killed the target and the other fuzed on the same target after it was dead.

We wish to form one vector of AKPF's that accounts for the Firings associated with each AKPF. As an intermediate step, Table 4.7 shows the results of repeating the AKPF in each replication a number of times equal to Firings in that replication.

Table 4.7. Forming an AKPF Vector

Replication Number	Scenario X	
	Firings	AKPF
1	1	0.50
1	1	0.50
3	1	1.00
3	1	1.00
4	1	0.25
4	1	0.25
4	1	0.25
4	1	0.25
5	1	0.00
5	1	0.00

Note that since replication 2 had no Firings, its AKPF of zero is not entered into the new AKPF vector. Finally, we display the vector of AKPF's that we may bootstrap from in Table 4.8.

Table 4.8. AKPF Vector for Bootstrap Sampling

AKPF
0.50
0.50
1.00
1.00
0.25
0.25
0.25
0.25
0.00
0.00

The process of constructing an AKPF vector is summarized in Figure 4.4.

BRAWLER Data

Replication	Scenario X		
Number	Kills	Firings	AKPF
1	1	2	0.50
2	0	0	0.00
3	1	2	1.00
4	1	4	0.25
5	0	2	0.00

Form an equal weight vector

Replication	Scenario X	
Number	AKPF	Firings
1	0.50	1
1	0.50	1
3	1.00	1
3	1.00	1
4	0.25	1
4	0.25	1
4	0.25	1
4	0.25	1
5	0.00	1
5	0.00	1

Bootstrap from this vector

Figure 4.4. Creating an AKPF Vector for Bootstrapping

As in Johnson and Lucas' program, we construct 501 bootstrap samples, which are all AKPF vectors of size $\sum_{i=1}^{n} Firings_i$, which we abbreviate as ΣF. Once we have 501 bootstrap samples of size ΣF, we take the following steps, which are similar to the steps followed in *enoughruns.*

1. Form 501 bootstrap samples of size ΣF from the AKPF vector.
2. Calculate the mean AKPF, AKPF$^*(b)$, of each of the 501 bootstrap samples.
3. Sort the 501 AKPF$^*(b)$'s in ascending order.
4. Calculate the following statistics for the 501 AKPF$^*(b)$'s:

   a. The median, or 50th percentile, AKPF$^*(b)^{(.50)}$, which is the 251st ordered value of the sorted AKPF$^*(b)$'s,

   b. The 5th percentile, AKPF$^*(b)^{(.05)}$, which is the 26th ordered value, and

   c. The 95th percentile, AKPF$^*(b)^{(.95)}$, which is the 476th ordered value.

5. Stop performing runs if both AKPF$^*(b)^{(.05)}$ and AKPF$^*(b)^{(.95)}$ are within $\pm 10\%$ of AKPF$^*(b)^{(.50)}$. Otherwise continue.

4.6.5. Preparing BRAWLER Data for Candidate S2

In order to apply Candidate S2, consider how must obtain the AKPF from each individual BRAWLER run. We do this by running Taranto's *STATS* post-processing program on each individual BRAWLER log file. However, *STATS* is designed to calculate the mean AKPF for all BRAWLER replications that have been performed, and it automatically and sequentially processes all the BRAWLER log files that appear in a working directory and then returns the mean AKPF. So, in order to obtain the AKPF for each individual log file in a directory instead of all the runs considered *en masse*, we created a simple *Unix C-Shell* program called *make_flyAKPF* , found in Appendix F. This program calculates the AKPF values for each run individually using *STATS* and creates one output file of AKPF and Firings from each run. This file is easily transformed into an $N \times 2$ matrix for use with S2. The process for preparing BRAWLER AKPF data for use by S2 is displayed in Figure 4.5.

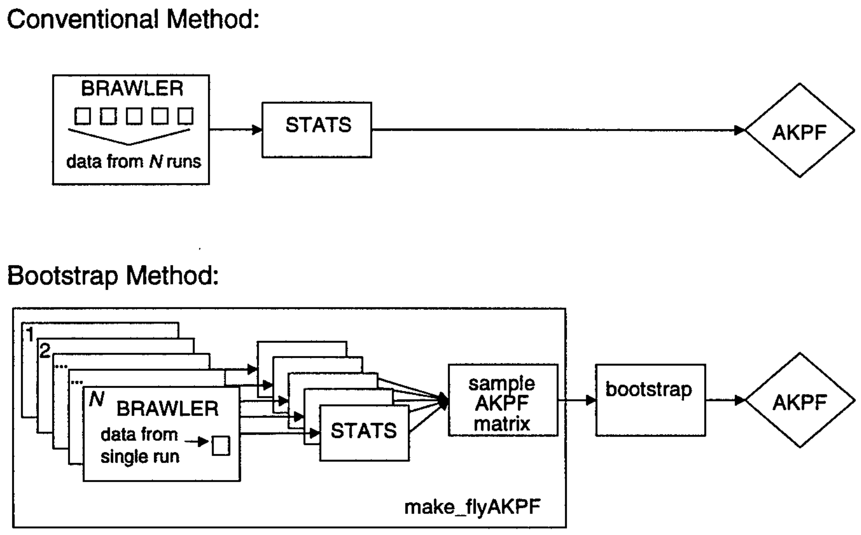


Figure 4.5. Preparing BRAWLER AKPF Data for Bootstrapping

5. Results

5.1. Introduction

Recall that the main purpose of this research is to investigate if the number of BRAWLER replications necessary to obtain a bootstrap confidence interval that meets our stopping criterion is less than the number required to obtain a standard confidence interval meeting the same criterion. By setting $\alpha = .10$ and $\gamma = .10$, our stopping criteria is as follows:

- Stop performing replications when a 90% confidence interval about the estimated mean can be constructed such that both endpoints of the interval are within $\pm 10\%$ of the mean, or in other words, the 90% confidence interval satisfies the requirement that the relative error of the mean is .10.

The values $\alpha = .10$ and $\gamma = .10$ were chosen primarily because they are the same values that are used in *enoughruns*. In accordance with the objective of the investigation, the following steps were performed for each of the three scenarios.

1. 200 BRAWLER replications were performed for each scenario and the 90% confidence intervals for the mean AKPF were calculated. These confidence intervals are our Truth model .

2. The minimum number N_1 of BRAWLER runs necessary to provide a 90% standard confidence interval about the estimated mean with a relative error of .10 for the mean AKPF was found for each scenario using Candidate S1..

3. By bootstrapping the AKPF data *B* times, where *B* = 501, the minimum number N_2 of BRAWLER replications necessary to provide a bootstrap 90% percentile interval that meets the same criterion as in part 2 was found for each scenario using Candidate S2.

4. N_1 and N_2, along with the two intervals generated by each method were compared for each replication of each scenario. The following questions were answered:

a. Is E(N_2) < E(N_1)? In other words, can a 90% bootstrap percentile interval using N_2 replications, where $N_2 < N_1$ be constructed that meets the same stopping criterion as a standard 90% confidence interval using N_1 replications?

b. Do either (or both) of the two intervals calculated in steps 2 and 3 overlap the interval calculated in step 1? In other words, were the estimates statistically similar?

An exception to step 1 was Scenario 2, in which BRAWLER aborted an excessive number of replications for various reasons so that only 193 replications were performed. The AKPF and Firings data for each run appears in Appendix G. The results from step 1 appear in Table 5.1, where CI Low and CI Up are the lower and upper bounds of the 90% confidence interval, respectively. The results from steps 2 and 3 are presented in Sections 5.3 through 5.5.

Table 5.1. Truth Models Resulting from 200 BRAWLER Runs

Scenario	# Runs	CI Low	Mean	CI Up	CI Length
1	200	30.07	32.18	34.28	4.21
2	193	29.49	31.54	33.59	4.09
3	200	26.01	28.49	30.97	4.96

5.2. Candidate Measures of Merit

In this section we discuss certain properties that we deemed appropriate to assess the merit of each candidate. We call such a property a *measure of merit* (MOM). The two MOMs that we chose to evaluate Candidate *i, i* = 1,2 were as follows:

1. E(N_i): The expected value of the number of runs (N_i) required to meet our stopping criterion.

This MOM is straightforward. The smaller the value of E(N_i), the better, since the more runs we must perform, the more time and computer resources we must use.

2. Probability of Overlap (P_i): This probability is estimated by the proportion of time the confidence intervals constructed using Candidate *i* and the confidence interval constructed from our Truth model overlapped.

When confidence intervals overlap in this way we conclude that the results obtained from using both methods are statistically similar. However, if the intervals do not overlap, we conclude that the results are statistically different.

Given these facts, we wish to avoid the event that a candidate returns a confidence interval that meets the stopping criterion but does not overlap the confidence interval constructed from our Truth model . If this event were to occur, the interval returned would be deceptive because if the interval does not overlap the interval from our Truth model , it most likely does not cover the expected AKPF. Therefore, we define a binary variable Overlap. If the interval constructed by each candidate for each scenario overlaps the similarly constructed interval for our Truth model , we assign Overlap the value of one, otherwise, it is assigned zero (1 = overlap of intervals, 0 = no overlap). Then, given repeated application of our candidate procedures, such as we have done in Section 5.5, the proportion of time that Overlap = 1 occurred is an estimate of P_i.

5.3. Candidate Results: One Example

In this section, we compare the results from one example of using Candidate S1 and Candidate S2, using data from Scenario 1 (AAAA). When a candidate determines that the stopping criterion has been met, we say that the candidate issues a *stop order.* Figure 5.1 presents a visual display of the results of using S1, while Table 5.2 gives a summary of S1's performance. Likewise, Figure 5.2 presents a similar visual display for a single realization of using S2 on the same data and Table 5.3 presents a summary of S2's performance. Since S2 involves the stochastic bootstrap sampling procedure, this realization may vary each time the procedure is performed. The particular realization of S2 that appears in Figure 5.2 and Table 5.3 was chosen for display because in this realization, $N_2 \approx E(N_2)$. Specifically, $N_2 = 97$ in the chosen realization, and our estimate of $E(N_2)$ is 99.4. The variability of S2 when performed on Scenarios 1-3 is assessed in Section 5.4.

In both Figure 5.1 and Figure 5.2, when both confidence interval bands cross the relative error threshold bands, the candidate issues a stop order. From these figures, we observe that there are two subtle differences in these processes, which are noted below:

1. The process used by both candidates requires that both the upper and lower confidence bands be within the relative error threshold bands before determining that enough runs have been made. In Candidate S1's case, the confidence intervals are symmetric; therefore, on any given run, either neither or both of the confidence bands are within the relative error threshold bands. This is not the case with Candidate S2. Since bootstrap percentile intervals are not necessarily symmetric, on any given run, zero, one, or two confidence bands may be within the relative error threshold bands.

2. If we allow the BRAWLER runs to continue after a stop order is issued, we see that Candidate S1 is more consistent in issuing another stop order on the next run after the first stop order was issued. S1 will only reverse its decision to issue a stop order if the variance of the sample data increased sufficiently (due to the new data) from the value calculated on the previous run. In contrast, in a similar situation it is quite possible that S2 will reverse its decision to issue a stop order on the next run even if the variance of the sample data did not increase. This phenomenon is due to the stochastic nature of the bootstrap sampling process.

Figure 5.1 and Figure 5.2 allow us to compare, for each run, the lower and upper thresholds that meet our stopping criterion to the lower and upper endpoints of our confidence interval.

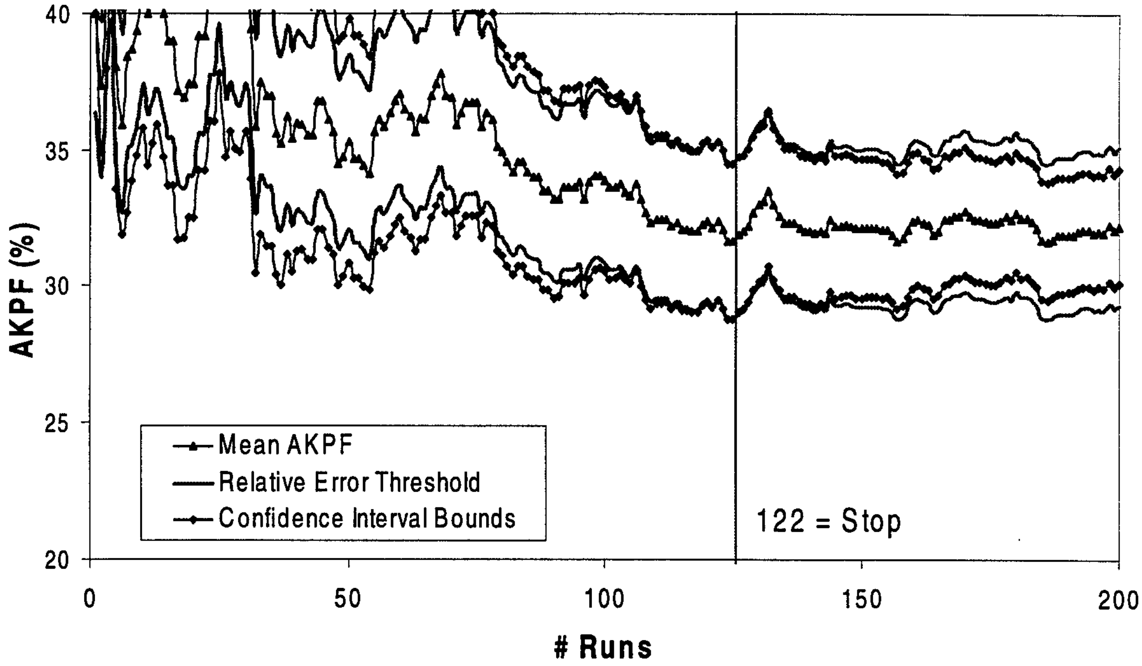


Figure 5.1. Candidate S1, Scenario 1

Table 5.2. Summary of Candidate S1 Performance, Scenario 1

# Runs to Stop (N_1)	122
Estimated Mean (E(AKPF$_1$))	32.39
90% Standard CI	(29.45, 35.32)
CI Length	5.87
Overlap	1

Using Candidate S1, the stopping criterion was met after 122 runs were performed. At 122 runs, the point estimator for the mean was 32.39 and the associated 90% standard confidence interval was (29.45, 35.32), which has a length of 5.87. The standard confidence interval from 122 runs overlaps the Truth model , the 90% confidence interval at 200 runs, which is (30.07, 34.28).

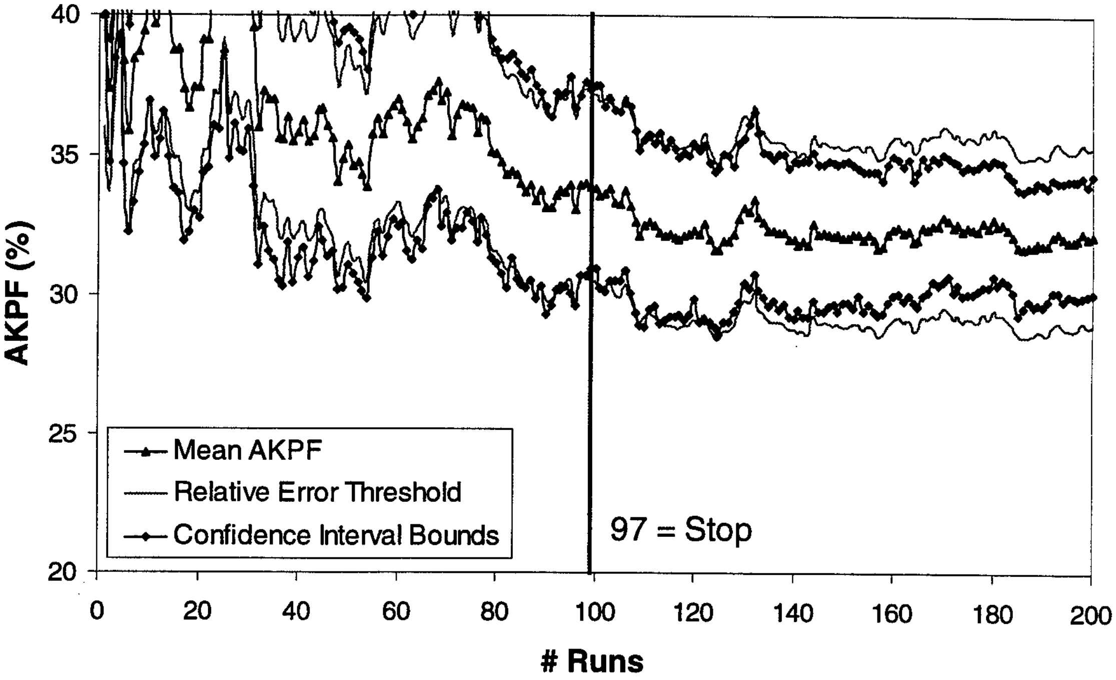


Figure 5.2. Candidate S2, Scenario 1

Table 5.3. Summary of Candidate S2 Performance, Scenario 1

# Runs to Stop (N_2)	97
Estimated Mean (E(AKPF$_2$))	33.92
90% Standard CI	(30.67, 37.13)
CI Length	6.47
Overlap	1

Using Candidate S2, the stopping criteria for the particular realization shown in Figure 5.2 was met after 97 runs had been performed. At 97 runs, the point estimator for the mean was 33.92 and the associated 90% bootstrap percentile interval was (30.67, 37.13), which has a length of 6.47. The bootstrap percentile interval from 97 runs also overlaps the Truth model.

5.4. Variability of Candidate S2

We have seen that the bootstrap method itself is a stochastic simulation process. Because of this, each time we use Candidate S2, we may obtain a different answer for both the number of runs required to meet the stopping criterion and the value of AKPF that is returned. Table 5.4 displays the results from 20 replications of S2 on each of our three scenarios. To aid making a comparison, Table 5.5 shows the similar results from using Candidate S1.

Table 5.4. Results from 20 Replications of Candidate S2

Candidate S2	Scenario 1		Scenario 2		Scenario 3	
Replication	N_2	AKPF	N_2	AKPF	N_2	AKPF
1	103	33.78	100	26.93	123	28.80
2	98	33.95	88	27.81	120	28.22
3	100	33.80	86	27.93	137	29.56
4	106	33.70	97	27.36	131	29.61
5	99	34.04	96	27.18	118	27.65
6	99	34.14	86	27.97	139	30.19
7	102	33.84	93	27.44	134	29.48
8	100	33.84	99	26.92	127	29.25
9	102	33.63	90	27.32	136	29.15
10	95	33.69	100	26.96	129	28.92
11	100	33.69	84	27.28	125	28.94
12	83	34.40	84	27.25	123	28.79
13	106	33.70	100	26.85	125	29.19
14	102	33.51	93	27.49	129	29.08
15	102	33.73	84	27.29	126	29.60
16	96	33.14	92	27.36	123	28.66
17	95	33.80	99	26.84	121	28.60
18	99	33.99	96	27.09	131	29.50
19	99	34.15	103	27.34	127	29.33
20	102	33.46	100	26.88	125	28.86
Expected Value	99.4	33.80	93.5	27.27	127.45	29.07

Table 5.5. Candidate S1 Results

Candidate S1	Scenario 1		Scenario 2		Scenario 3	
	N_1	AKPF	N_1	AKPF	N_1	AKPF
Value	122	32.39	117	27.85	190	28.53

We see that on average, Candidate S2 required 18.5% less runs than Candidate S1 to issue a stopping order on Scenario 1, 20.1% less on Scenario 2, and 32.9% on Scenario 3. These results are displayed in Table 5.6.

Table 5.6. Results of S1 and S2 on Scenarios 1-3

Candidate S*i*	Scenario 1		Scenario 2		Scenario 3	
	E(N_i)	E($AKPF_i$)	E(N_i)	E($AKPF_i$)	E(N_i)	E($AKPF_i$)
$i = 1$	122.0	32.39	117.0	27.85	190.0	28.53
$i = 2$	99.4	33.80	93.5	27.27	127.5	29.07
% Runs Saved with S2	18.5	-	20.1	-	32.9	-

5.5. Effect of the Order of BRAWLER Data

5.5.1. Introduction

Since both candidates are sequential procedures, the order in which the BRAWLER data is obtained affects both when a stop order is issued and the estimate of expected AKPF that is returned. When we mention the order in which the data was obtained, we are referring to the replication number on which a specific (multivariate) BRAWLER data is obtained. In our case, we are interested in the bivariate data consisting of APKF and Firings values. If for some reason the data obtained before a stop order was issued is biased, our sequential candidate procedures return biased estimates of the expected AKPF. Therefore, we determined it is necessary to investigate the performance of each candidate on different random orderings of the AKPF data obtained from our 200 BRAWLER runs.

5.5.2. Results of Investigation

We performed both candidate procedures on 20 different randomized orderings of the data from 200 BRAWLER runs for Scenario 1. Each time we perform a procedure on a different ordering of the data we say that we are performing a *metareplication* of the procedure. Although we performed 20 replications of Candidate S2 in the experiment mentioned in Section 5.4, in this experiment we performed only a single metareplication of S2 for each of the 20 orderings of the data.

Table 5.6 summarizes the performance of each of our candidates on the different metareplications. In its top section, the table shows the 90% confidence interval that was constructed for our Truth model for comparison purposes. In the lower section, the table lists data on the performance of each candidate. The table's first column lists the labels for the different orderings of BRAWLER data that were used. These include the original order of the data and the 20 reorderings (denoted Reorder *r*, where $r \in [1,20]$). In the successive columns, the following data is displayed for Candidate *i*, first for $i = 1$ and then for $i = 2$:

1. The number of runs N_i (denoted in the table as #Runs) required before Candidate *i* issued a stop order,
2. The AKPF value returned by Candidate *i* after N_i runs,
3. The associated 90% confidence interval (CI) for the AKPF in number 2 above, which, in the case of Candidate S2 is the 90% bootstrap percentile interval (PI).
4. The value of Overlap, as defined in Section 5.2.

Table 5.6. Candidate Performance on Different Orderings of BRAWLER Data

	#Runs	AKPF	90% CI for AKPF		Overlap	#Runs	AKPF	90% PI for AKPF		Overlap
	Truth Model									
	200	32.18	30.071	34.284	-	-	-	-	-	-
	Candidate 1					Candidate 2 - Single Realization				
Original Order	122	32.39	29.45	35.32	1	97	33.92	30.667	37.13	1
Reorder 1	119	30.86	28.07	33.64	1	73	31.27	28.191	34.36	1
Reorder 2	120	31.07	28.26	33.88	1	85	32.76	29.133	35.95	1
Reorder 3	117	31.49	28.63	34.35	1	87	30.61	27.886	33.36	1
Reorder 4	120	33.85	30.79	36.90	1	94	34.71	31.402	37.79	1
Reorder 5	115	33.81	30.75	36.87	1	99	33.67	30.527	36.92	1
Reorder 6	119	30.14	27.41	32.86	1	102	28.89	26.046	31.75	1
Reorder 7	140	30.21	27.47	32.96	1	94	29.54	26.778	32.22	1
Reorder 8	106	35.82	32.58	39.06	1	91	36.17	32.784	39.34	1
Reorder 9	86	33.46	30.43	36.48	1	75	32.07	29.091	35.24	1
Reorder 10	119	29.58	26.90	32.25	1	62	33.82	30.476	36.94	1
Reorder 11	138	33.08	30.09	36.07	1	122	33.31	30.023	36.60	1
Reorder 12	98	33.02	30.02	36.02	1	34	42.00	37.865	46.11	0
Reorder 13	105	32.60	29.67	35.53	1	83	32.30	29.742	35.37	1
Reorder 14	141	30.96	28.15	33.77	1	111	30.59	28.004	33.46	1
Reorder 15	122	33.32	30.30	36.34	1	95	32.76	29.552	35.96	1
Reorder 16	129	32.29	29.37	35.22	1	97	30.85	27.83	33.55	1
Reorder 17	100	32.42	29.53	35.30	1	78	31.35	28.417	34.45	1
Reorder 18	101	30.33	27.62	33.05	1	81	28.45	25.603	31.28	1
Reorder 19	106	32.28	29.37	35.19	1	84	31.41	28.464	34.46	1
Reorder 20	121	31.55	28.73	34.37	1	106	30.15	27.145	33.16	1
Mean Value	**116.38**	**32.12**	-	-	-	**88.10**	**32.41**	-	-	-
Overlap Prop	-	-	-	-	**1.00**	-	-	-	-	**0.95**

5.5.3. Expected Number of Runs

We constructed approximate 90% standard confidence intervals for the estimated means of the number of runs N_1 and N_2 required before a stop order was issued. These intervals appear in Figure 5.1below. From this figure we may infer that $E(N_2) < E(N_1)$; however, a formal comparison is made in Section 5.5.5. Our estimate of $E(N_1)$ based on the sample data is 116.38, which means that we would expect to perform approximately

117 runs of Scenario 1 before meeting our stopping criterion using Candidate S1. Furthermore, we state with 90% confidence that E(N_1) lies in the interval (111.04, 121.73). In contrast, our estimate of E(N_2) is 88.10, so we would expect to perform approximately 89 runs of Scenario 1 before stopping when using Candidate S2, and we state with 90% confidence that E(N_2) lies in the interval (81.15, 95.04). So, based on 20 metareplications of Scenario 1, we see that on average S2 issued a stopping order using 24% fewer runs than S1.

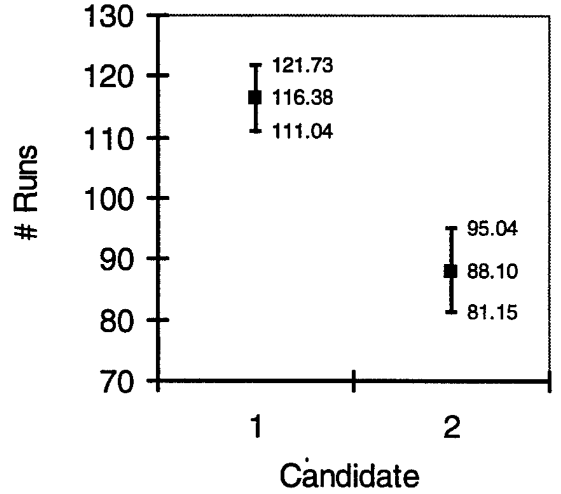


Figure 5.1. 90% Confidence Intervals for E(N_1) and E(N_2)

5.5.4. Expected Value of AKPF

Next, we constructed approximate 90% standard confidence intervals for the estimated mean of $AKPF_1$ (the AKPF resulting from a stop order issued by S1 at run N_1), and $AKPF_2$ (the AKPF for S2 at run N_2). These intervals appear in Figure 5.2. From this figure we may infer that E($AKPF_1$) = E($AKPF_2$); however, the formal comparison appears in Section 5.5.5. Our point estimate of E($AKPF_1$) based on the sample data is 32.12.

sample data is 32.12. Furthermore, we state with 90% confidence that E($AKPF_1$) for Scenario 1 (when our stopping criterion is met using S1) lies in the interval (31.54, 32.70). Our point estimate of E($AKPF_2$) is 32.41. we can state with 90% confidence that E($AKPF_2$) for Scenario 1 (when our stopping criterion is met using S2) lies in the interval (31.30, 33.51). The confidence interval for S2 is wider than that for S1 due to the variability introduced by the stochastic nature of the bootstrap, as discussed in Section 5.4. For comparison purposes, Figure 5.3 displays the 90% confidence interval constructed from our Truth model . This figure appears here to show that both point estimates for E($AKPF_1$) and E($AKPF_2$) compare well with the point estimate of the expected AKPF from our Truth model . However, *the reader should not directly compare the length of the intervals in Figure 5.2 and Figure 5.3.* The interval in Figure 5.3 is used to estimate the expected AKPF of Scenario 1 based on a sample size of 200 BRAWLER replications. In contrast,the intervals in Figure 5.2 are used to estimate the mean AKPF response for Scenario 1 that is obtained by each candidate based on a sample size of 20 metareplications of the 200 BRAWLER replications.

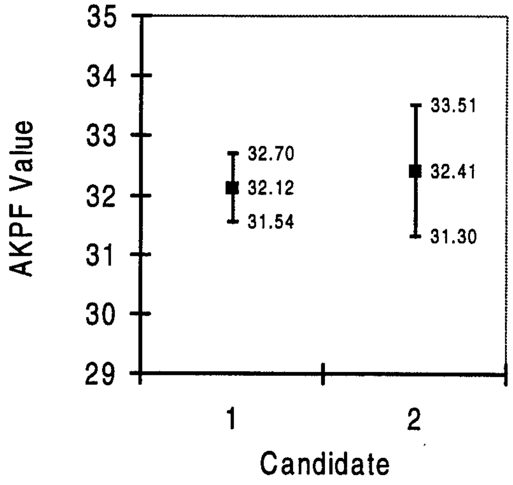
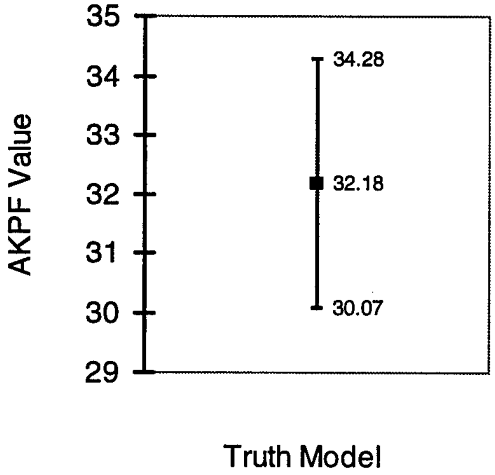


Figure 5.2, 90% Confidence Intervals for E($AKPF_1$) and E($AKPF_2$)



Note*:* We cannot directly compare the length of the intervals in Figure 5.2 and Figure 5.3. See text above for an explanation.

Figure 5.3. 90% Confidence Interval for E(*AKPF*) from Truth Model

5.5.5. Paired Comparisons

We have been comparing the two candidates on the basis of two different measures, namely:

1. The number of runs required before a stop order is issued (a MOM), and
2. The value of AKPF returned and its associated confidence interval.

We may make these comparisons by constructing a 90% confidence interval for the difference in the expected values in numbers 1 and 2 above for each candidate, which we may write explicitly as:

3. $\mathrm{E}(N_1) - \mathrm{E}(N_2)$
4. $\mathrm{E}(AKPF_1) - \mathrm{E}(AKPF_2)$

Since we can pair the observations in Table 5.6 according to the ordering of the data they were subjected to, we may make a paired comparison to estimate the values in numbered items 3 and 4 above. We may do this by constructing a *paired-t confidence interval* for the difference in the expected values we are interested in. A 90% paired-*t* confidence interval for $\mathrm{E}(N_1) - \mathrm{E}(N_2)$ appears in Figure 5.4. In this figure we can see that the difference in expected values of N_1 and N_2 is significant at $\alpha = .10$ since the confidence interval does not include zero. The interval displayed is (22.91, 33.66) and the point estimator of $\mathrm{E}(N_1) - \mathrm{E}(N_2)$ is 28.29.

Likewise, a paired-t 90% confidence interval for $\mathrm{E}(AKPF_1) - \mathrm{E}(AKPF_2)$appears in Figure 5.5. In contrast to Figure 5.4, Figure 5.5 shows that the difference in expected values of $AKPF_1$ and $AKPF_2$ is not significant at $\alpha = .10$ since the portrayed confidence interval includes zero. The interval displayed is (-1.20, 0.62) and the point estimator of $\mathrm{E}(AKPF_1) - \mathrm{E}(AKPF_2)$ is -0.29.

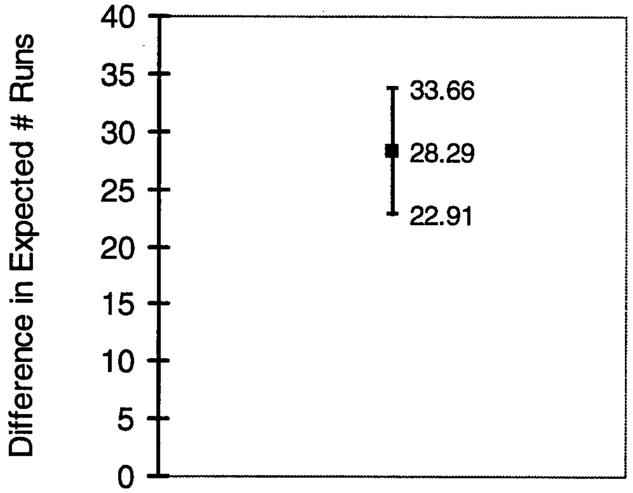


Figure 5.4. 90% Paired-t Confidence Interval for E(N_1) - E(N_2)

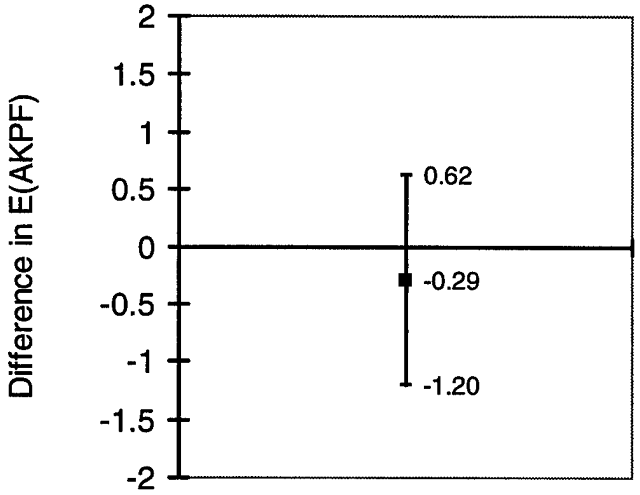


Figure 5.5. 90% Paired-t Confidence Interval for E($AKPF_1$) - E($AKPF_2$)

5.5.6. Probability of Overlap

As can be seen in Table 5.6, the proportion of time the confidence intervals constructed using Candidate *i* and the confidence interval constructed from our Truth model overlapped was 1.00 for $i = 1$ and 0.95 for $i = 2$. We use these values to estimate P_1 and P_2, respectively. So, based on our sample of 20 metareplications, we estimate the value of P_1 to be 1.00 and P_2 to be 0.95. We constructed a 90% confidence interval for $E(P_1 - P_2)$ which appears in Figure 5.6. Figure 5.6 shows that the expected value of $P_1 - P_2$ is not significant at $\alpha = .10$ since the 90% confidence interval includes zero. The interval displayed is (-0.03, 0.13) and the point estimator of $P_1 - P_2$ is 0.05.

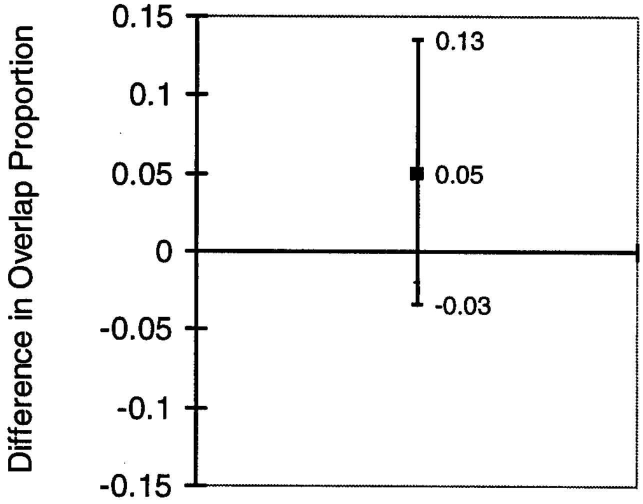


Figure 5.6. 90% Confidence Interval for $E(P_1 - P_2)$

6. Conclusions

6.1. Summary

We have demonstrated proof of the SIMAF concept through our pilot program that demonstrates the cross-resolution modeling process through the calibration of the campaign-level model THUNDER with the engagement-level model BRAWLER. Since both of these models are recommended members of the USAF Analysis Toolkit, they will continue to be used for several years while the USAF transitions to newer models. Using cross-resolution modeling, we can ensure that outputs from THUNDER are consistent with their BRAWLER counterparts. We can also ensure that a pedigree is established for data crossing levels of model resolution. We have shown how a particular MOE from BRAWLER, the adjusted kills per firing (AKPF), is transformed into input data for THUNDER air-to-air probability of kill (PK). The procedures in Chapter 5 are different ways of estimating the expected value of the AKPF in BRAWLER, which is the value that is used as the THUNDER air-to-air PK. All of the procedures presented in Chapter 5 provided sufficiently accurate estimates of E(*AKPF*).

Using sequential procedures, we can minimize the number of BRAWLER runs necessary to provide us with a sufficiently accurate estimate of E(*AKPF*). We define sufficient accuracy up front in terms of a relative error ε and significance level α, which determines our stopping criterion. A $(1-\alpha)\%$ confidence interval may be automatically constructed after each run that lets us determine if we have met our stopping criterion.

Minimizing the number of runs in this way allows us to increase the speed of the THUNDER air-to-air calibration process.

In particular, we found that the sequential Candidate S1, which uses the bootstrap, performs better than the sequential Candidate S2, which uses standard normal theory. In our 20 metareplications of Scenario 1, we found that on average S2 issued a stopping order using 24% fewer runs than S1, while returning statistically similar results for both the expected value of AKPF and the probability of overlapping the confidence interval from our Truth model . However, both procedures perform well, so they are both recommended for SIMAF use.

6.2. Discussion

It should be noted that bootstrap methods are an alternative to standard statistical procedures, not a replacement for them (Cheng, 1995). Bootstrap methods used in conjunction with classical statistical procedures provide both the analyst and the decision-maker a great deal of insight about the distribution of a statistic of interest.

Through the use of bootstrap sampling, we found that a major asset of the bootstrap is its ability to build up a picture of the distribution of a statistic of interest based on a sample. This is especially valuable for smaller sample sizes where a normality assumption may not be appropriate. Not only is the bootstrap a powerful tool, but it can be implemented easily and quickly by performing a fairly trivial computer simulation. The simulation can be written in any number of programming languages – general programming languages such as *C++* or *FORTRAN*, or mathematical programming languages are examples.

With modern mathematical programming languages, such as *Mathcad*, the process of building a picture of the distribution of a statistic can also be graphically animated. Using the *Mathcad 6.0* program we created, we were able to do this quite effectively. However, it took a longer time to learn how to exploit this capability than it did to implement the actual bootstrap procedure. Yet this capability is especially valuable in this day and age where emphasis on the visual display of data is ever increasing. Data visualization techniques are a key part of the SIMAF concept, therefore the bootstrap is a very appropriate tool for SIMAF use.

6.3. Additional Research

Another useful tool for the simulation analyst is a between-run prediction of the number of runs that must still be performed (runs to go) in order to meet the chosen stopping criterion. Some initial experimentation in this area was accomplished in this research, but due to time limitations was not fully explored; therefore, only initial results are presented here. We considered two candidates P1 and P2 for predicting the number of runs to go. These candidates are closely related to candidates S1 and S2 for determining if our stopping criterion was met.

6.3.1. Candidate P1

Candidate P1 makes use of another procedure presented by Law and Kelton (1991) in order to make such a prediction using the standard confidence interval. We combined this procedure with Law and Kelton's sequential procedure described in Section 4.6.2. Using this combination, we were able to obtain a prediction of the number of runs yet to be performed after every run that did not meet the stopping criterion.

Law and Kelton present an approximate expression for the number of replications needed to obtain a relative error γ, given by:

$$n_r^*(\gamma) = \min\left\{ i \geq n : \frac{t_{(i-1,1-\alpha/2)} \cdot \sqrt{\hat{\sigma}^2(n)/i}}{|\bar{x}(n)|} \leq \gamma' \right\} \tag{6.1}$$

where $t_{(i-1,1-\alpha/2)} \cdot \sqrt{\hat{\sigma}^2(n)/i} = hl(i,\alpha)$ is the same confidence interval half-length as in (4.21) except that it is now based on $i \geq n$ observations instead of n. We make an assumption that our estimates of the population mean and population variance, $\bar{x}(n) = \hat{\theta}(n)$ and $\hat{\sigma}^2(n)$, respectively, do not change as more replications are performed. Then $n_r^*(\gamma)$ is approximated by the smallest integer i that satisfies $i \geq \hat{\sigma}^2(n)[z_{(1-\alpha/2)}/\gamma' \cdot \bar{x}(n)]^2$. Using (6.1), we were able to obtain a prediction of the number of runs yet to be performed after every run that did not meet the stopping criterion. We decided that the best way to judge the performance of this procedure was to compare the following:

1. A plot of the predicted number of runs to go versus the current run number.
2. A plot of the actual number of runs to go versus the current run number. This is a straight line we denote as the Truth Line.

Based on a visual inspection of these plots, we conclude that this procedure performs well on our three scenarios. More formal methods for judging performance, such as calculating a mean squared error based on the deviations from the Truth Line could be implemented if so desired. Our results are presented in Figure 6.1 through Figure 6.3. Since we use a set of 20 pilot runs to obtain an initial prediction, predictions for runs

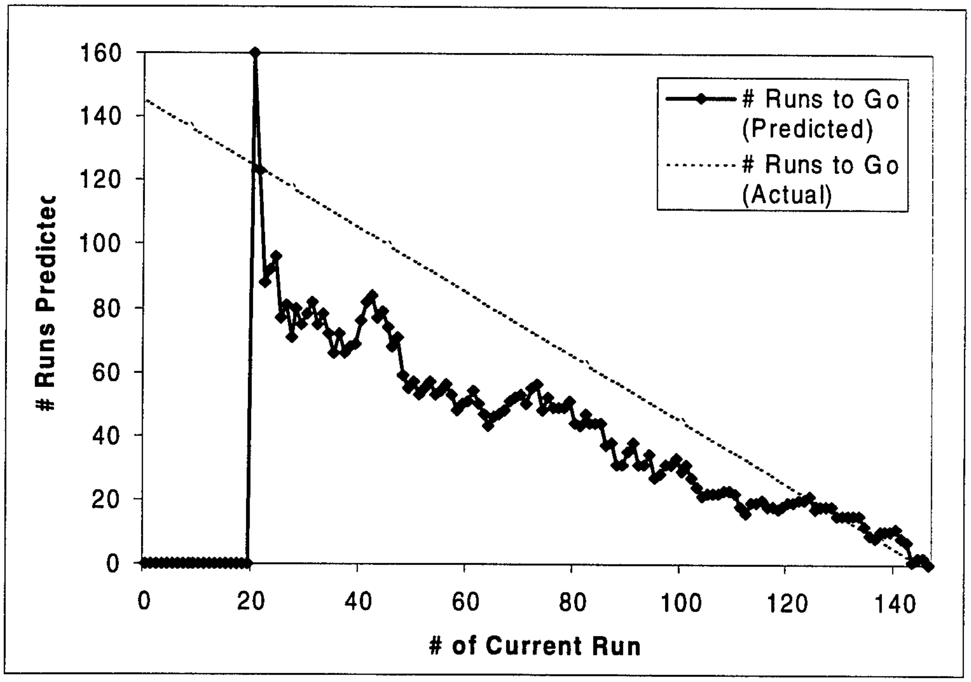


Figure 6.1. Predictions of Runs to Go for Scenario 1 using P1

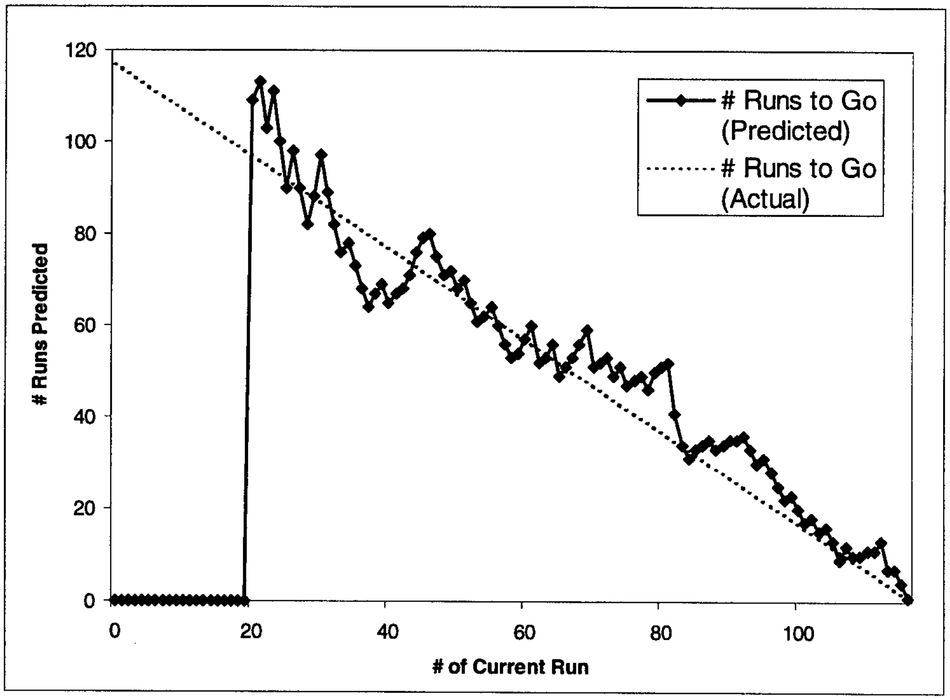


Figure 6.2. Predictions of Runs to Go for Scenario 2 using P1

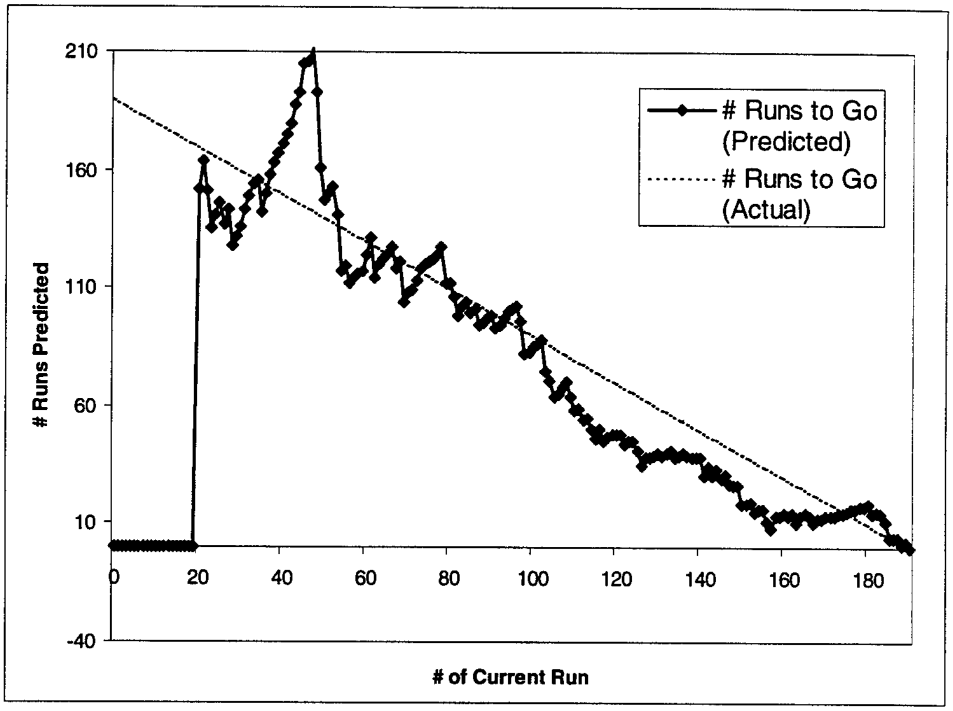


Figure 6.3. Predictions of Runs to Go for Scenario 3 using P1

1-19 are not made, so for each of these runs the predicted number of runs to go is assigned a value of zero.

6.3.2. Candidate P2

The bootstrap procedure is normally involved with generating bootstrap samples and then obtaining bootstrap statistics for each of these bootstrap samples. A key aspect of the bootstrap process is this: Given a statistic of interest for a sample, such as the mean, a large sample of B bootstrap statistics has nearly the same variance as the original statistic of interest. Therefore, the variance of the bootstrap statistics can be used to estimate the variance of the original statistic (Cheng, 1995).

We estimate a standard error (standard deviation of the sample mean) with the standard deviation of a large sample of B bootstrap means, denoted $\hat{\sigma}_{\hat{\theta}^*}(B)$. We write this as $\sqrt{\hat{\sigma}^2(n)/n} \approx \hat{\sigma}_{\hat{\theta}^*}(B)$. The bootstrap estimate of standard error is

$$\hat{\sigma}_{\hat{\theta}^*}(B) = \left\{ \sum_{b=1}^{B} \left[\hat{\theta}^*(b) - \hat{\theta}^* \right] \Big/ (B-1) \right\}^{\frac{1}{2}} \tag{6.2}$$

where $\hat{\theta}^*(b)$ is the estimate of $\hat{\theta}$ for the bootstrap sample x^{*b}, $\hat{\sigma}^*_{\hat{\theta}}$ is the estimated standard error of $\hat{\theta}^*$ for the bootstrap sample x^{*b} and $\hat{\theta}^* = (1/B) \cdot \sum_{b=1}^{B} \hat{\theta}^*(b)$.

We use the bootstrap estimate of standard error in place of the original estimate of standard error to create a slight variation on (6.1), as follows:

$$n_r^*(\gamma) = \min\left\{ i \geq n : \left[\frac{t_{(i-1,1-\alpha/2)} \cdot \left(\hat{\sigma}_{\hat{\theta}^*}(B) \cdot \sqrt{n} / \sqrt{i} \right)}{\left| \bar{x}(n) \right|} \right] \leq \gamma' \right\} \tag{6.3}$$

We then sequentially predict the number of runs to go after each run is completed using (6.3). The reason for using (6.3) instead of (6.1) is to investigate if using the bootstrap estimate of standard error yields different prediction results than using the standard estimate does. The results for P2 used on Scenario 1 are presented in Figure 6.4 for $B = 501$. The series denoted *Rep i,* where $i \in [1,10]$, plots the predictions of the number of runs to go on the y-axis against the number of runs completed on the x-axis for each of 10 replications of P2. The series denoted *Mean Predict to Go* plots the mean of the predictions made after each run is completed for 10 replications of P2. Finally, the series denoted *Mean Actual to Go* plots the actual number of runs to go, based on using run 121 as the final run (stopping run). Run 121 was chosen because for the 10 replications of P2, the mean number of runs to meet the stopping criterion is 121.2. Table 6.1 presents the results of 10 replications of P2 on Scenario 1, using the convention in Table 5.4.

Initial results seem to indicate that there is a difference between the predictions made by P1 and the mean of the predictions made by P2 on a given run. It appears that the mean of predictions made by P2 may perhaps be slightly more accurate than a prediction made by P1 on runs distant from the stopping run. However, the predictions made by P2 on a given run have a very high variance, most noticeably when the given run is close to the stopping run. Again, this is due to the stochastic nature of the bootstrap.

Therefore, based on our initial experimentation, a single prediction of the number of runs to go made by P2 using the bootstrap estimate of standard error does not appear to be useful. However, if the prediction process is replicated, the mean of the predictions made by P2 appears to be fairly accurate on runs far from the stopping run. Since P2's prediction process may be replicated quickly, P2 may be a useful tool for predicting the number of runs to go. The best strategy may be to rely on P2 for predictions when the current run is far from the stopping run, and then start relying on P1 when the predictions from P2 become sufficiently small (for example, 30 runs to go).

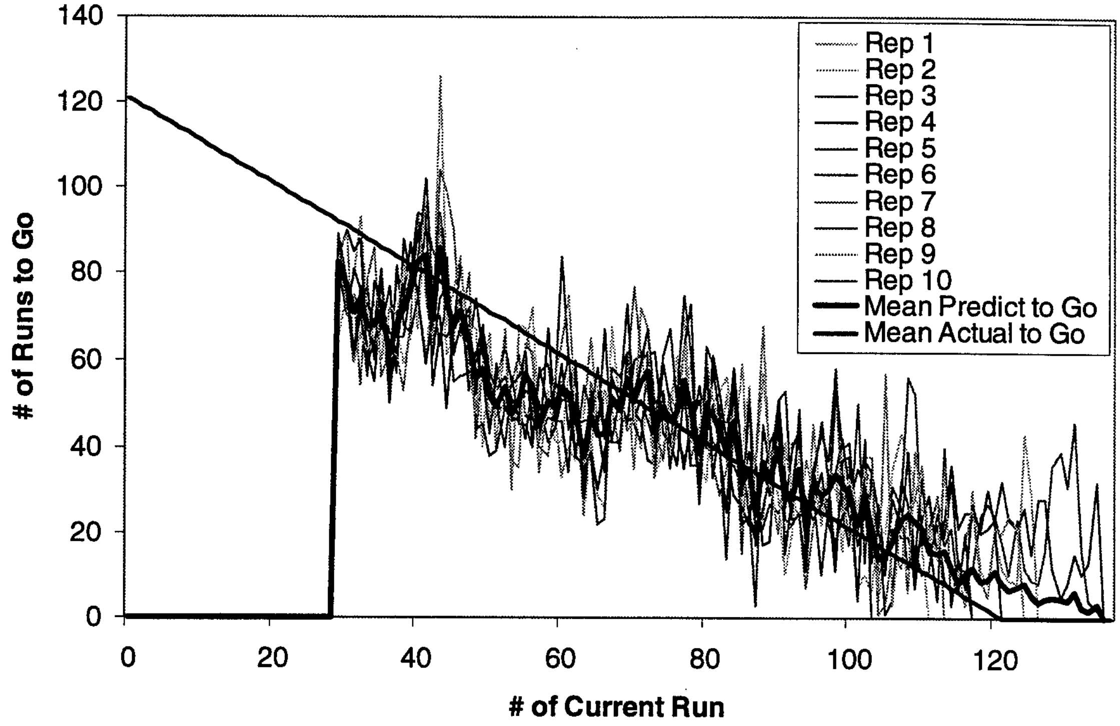


Figure 6.4. Predictions of Runs to Go for Scenario 1 using P2

Table 6.1. Results from 10 Replications of Candidate P2

Replication	N_2	AKPF
1	112	29.35
2	127	28.42
3	136	29.75
4	118	28.73
5	136	29.75
6	122	28.68
7	114	28.69
8	104	29.59
9	125	28.52
10	118	28.73
Mean	121.2	29.02

Appendix A. *bootstrap.mcd*

BOOTSTRAP SAMPLING PROGRAM using Mathcad 6.0

(Based on RAND Corp.'s *enoughruns.F* Fortran program designed
to determine when enough BRAWLER runs have been made)

GLOBAL PARAMETERS:

$B := 501$ B is the # of bootstrap reps. B can be any number you choose. Typically, B is chosen such that 500<B<2000. In *enoughruns.F*, B:=501

WARNING: INCREASING B INCREASES PROCESSING TIME EXPONENTIALLY

p and q are used to look at the advantages of using bootstrapping with less runs vs. normal methods with more runs.

$p := 94$ p is the number of data points in the output data vector (called Data below) that you would like to consider for bootstrapping (1<p<q<=k, where k is the total number of data points you have available). Typically, you would never set p<20 when using bootstrapping. In *enoughruns.F*, p>19, since 20 is the minimum number of BRAWLER runs that must be made before *enoughruns.F* begins bootstrapping runs.

$q := 117$ q is the number of data points in the output data vector (called Data below) that you would like to consider for non-bootstrapping (1<p<q<=k, where k is the total number of data points you have available).

$\alpha := .10$ α is the significance level $(0<\alpha<1)$

$\gamma := .10$ γ is the relative error $(0<\gamma<1)$

When we obtain an estimate of μ, the population mean, with a relative error of γ and a confidence level of $100(1-\alpha)\%$, we say that we have enough simulation data, or, in other words, "enough runs."

Notes: ORIGIN has been set to 1 in Menu Item Math - Built-In Variables
Also on Menu: Math - Automatic Mode must be enabled for animation to work

Function sumfirings counts the total # of firings in all the runs in X. sumfirings expectsX to be a matrix of AKPFs and Firings.

```
sumfirings(X) := | sum ← 0
                 | r ← rows(X)
                 | for  i ∈ 1..r
                 |    $sum \leftarrow sum + X_{i,2}$
                 | sum
```

Function weightPks creates an equally weighted vector (w.r.t. Firings) of AKPFs suitable for bootstrapping. The AKPF for a run is entered in the vector a number of times = to the # of Firings in that run

```
weightPks(X) := | s ← sumfirings(X)
                | r ← rows(X)
                | $W_1 \leftarrow 0$
                | for  i ∈ 1..r
                |   if $X_{i,2} > 0$
                |      | for  $j \in 1..X_{i,2}$
                |      |    $Wt_j \leftarrow X_{i,1}$
                |      | W ← stack(W, Wt)
                |      | Wt ← 0
                | W ← submatrix(W, 2, s+1, 1, 1)
                | W
```

Function bootstrap_1 bootstraps 1 value from the runs data (original sample). To obtain a bootstrap sample, the function must be replicated k times where k is the size of the original sample. It takes a vector X of size k and returns 1 randomly sampled element of X

```
bootstrap_1(X) := | r ← rows(X)
                  | N ← X
                  | u ← rnd(r)
                  | u ← ceil(u)
                  | $x \leftarrow N_u$
                  | x
```

Function bootstrap_1pair bootstraps 1 ordered pair (AKPF,Firings) from the runs data (original sample). To obtain a bootstrap sample, the function must be replicated N times where N is the size of the original sample. It takes a matrix X of size Nx2 and returns 1 randomly sampled ordered pair (AKPF,Firings)

```
bootstrap_1pair(X) := | r ← rows(X)
                      | N ← X
                      | u ← rnd(r)
                      | u ← ceil(u)
                      | $x \leftarrow \left(N^T\right)^{\langle u \rangle}$
                      | $x^T$
```

Function bootstrap_k bootstraps k values from an original sample of size k. It takes a vector X of size k and returns a randomly resampled (bootstrap)version of X.

```
bootstrap_k(X) := | r ← rows(X)
                  | N ← X
                  | for  i ∈ 1..r
                  |    | u ← rnd(r)
                  |    | u ← ceil(u)
                  |    | $X_i \leftarrow N_u$
                  | X
```

Function interval takes a data point x and a relative error γ and calculates the interval that is within the relative error of x.

```
interval(x,γ) := | $interval_1 \leftarrow x \cdot (1 - \gamma)$
                 | $interval_2 \leftarrow x \cdot (1 + \gamma)$
                 | interval
```

Function stdev calculates standard deviation. It takes a vector X and returns the standard deviation of X. In this function, the variance of X is modified from Mathcad's built in function var, since Mathcad uses the n divisor instead of n-1.

```
stdev(X) := | n ← rows(X)
            | sd ← $\sqrt{\mathrm{var}(X)\cdot\frac{n}{n-1}}$
            | sd
```

Function bootstrap_mean takes a vector X, forms one bootstrap sample from X and returns the mean of the bootstrap sample

```
bootmean(X) := | r ← rows(X)
               | for i ∈ 1 .. r
               |   $N_i$ ← bootstrap_1(X)
               | m ← mean(N)
               | m
```

Function CIhalflength takes a vector X and a significance level a and returns the half-length of a 100(1-α)% confidence interval

```
CIhalflength(X, α) := | n ← rows(X)
                      | se ← stderr(X)
                      | tvalue ← $\mathrm{qt}\left(1-\frac{\alpha}{2}, n-1\right)$
                      | CIhalflength ← tvalue·se
                      | CIhalflength
```

Function stderr calculates standard error. It takes a vector X and returns the standard error of X. In this function, the variance of X is modified from Mathcad's built in function var, since Mathcad uses the n divisor instead of n-1.

```
stderr(X) := | n ← rows(X)
             | varX ← $\mathrm{var}(X)\cdot\frac{n}{n-1}$
             | se ← $\sqrt{\frac{varX}{n}}$
             | se
```

Function bootstrap_mean takes a vector X, forms one bootstrap sample from X and returns the mean and standard error of the bootstrap sample

```
bootstrap_mean_se(X) := | r ← rows(X)
                        | for i ∈ 1 .. r
                        |   $N_i$ ← bootstrap_1(X)
                        | m ← mean(N)
                        | se ← stderr(N)
                        | $\begin{pmatrix} m \\ se \end{pmatrix}$
```

Function CIhalflength takes a vector X and a significance level a and returns a 100(1-α)% confidence interval for the mean of X

```
CI(X, α) := | mn ← mean(X)
            | hl ← CIhalflength(X, α)
            | $CI_1$ ← mn − hl
            | $CI_2$ ← mn + hl
            | CI
```

Function perc_interval takes a vector X and a significance level α, calculates the 5th and 95th percentile of X, and returns the interval having those percentiles as endpoints.

```
$perc\_interval(X,\alpha) :=$
    $r \leftarrow rows(X)$
    $Xs \leftarrow sort(X)$
    $Lo \leftarrow ceil\left(r \cdot \frac{\alpha}{2}\right)$
    $Up \leftarrow ceil\left[r \cdot \left(1 - \frac{\alpha}{2}\right)\right]$
    $perc\_interval_1 \leftarrow Xs_{Lo}$
    $perc\_interval_2 \leftarrow Xs_{Up}$
    $perc\_interval$
```

Function PIhalflength takes a vector X and a significance level α and returns the half-length of the percentile interval returned by function perc_interval

```
$PIhalflength(X,\alpha) :=$
    $int \leftarrow perc\_interval(X,\alpha)$
    $PIhalflength \leftarrow \frac{int_2 - int_1}{2}$
    $PIhalflength$
```

Function boot _CIs takes a matrix X of AKPF & Firings runs data, the number of bootstrap replications n, the significance level a, and a kludge factor. kludge should be set to the position of the first nonzero Firings entry in the X data matrix. This is a workaround so that if initial values of Firings are equal to zero the program will perform correctly. This function returns a set of bootstrap percentile intervals for X. The runs data is read in one run at a time and a percentile interval is calculated after each new run is added.

```
$boot\_CIs(X,n,\alpha,kludge) :=$
    $q \leftarrow rows(X)$
    $Xs \leftarrow csort(X,1)$
    $max \leftarrow Xs_{q,1}$
    for $j \in kludge..q$
        $Xj \leftarrow submatrix(X,1,j,1,2)$
        $Xjw \leftarrow weightPks(Xj)$
        for $i \in 1..n$
            $M_i \leftarrow bootmean(Xjw)$
        $Med_j \leftarrow median(M)$
        $PI^{<j>} \leftarrow perc\_interval(M,\alpha)$
    $\begin{bmatrix} Med \\ \left(PI^T\right)^{<1>} \\ \left(PI^T\right)^{<2>} \end{bmatrix}$
```

Function boot _means takes a matrix X of AKPF & Firings runs data, the number of bootstrap replications n, the significance level α, and a kludge factor. kludge should be set as in function boot_CIs. This function returns a set of n bootstrap means for X, where the bootstrap mean is the median of the means of each bootstrap sample. The runs data is read in one run at a time and a bootstrap mean is calculated after each new run is added.

```
$boot\_means(X,n,\alpha,kludge) :=$
    $q \leftarrow rows(X)$
    $Xs \leftarrow csort(X,1)$
    $max \leftarrow Xs_{q,1}$
    for $j \in kludge..q$
        $Xj \leftarrow submatrix(X,1,j,1,2)$
        $Xjw \leftarrow weightPks(Xj)$
        for $i \in 1..n$
            $M_i \leftarrow bootmean(Xjw)$
        $Med_j \leftarrow median(M)$
    $Med$
```

Function boot_Z takes a vector X and the number of bootstrap replications n and calculates the bootstrap version of the Z statistic for each element in X

```
$boot\_Z(X,n) :=$
    for $i \in 1..n$
        $\theta_i \leftarrow mean(X)$
        $se_i \leftarrow stderr(X)$
        $M^{\langle i \rangle} \leftarrow bootstrap\_mean\_se(X)$
        $Z_i \leftarrow \frac{M_{1,i} - \theta_i}{M_{2,i}}$
        $bound_i \leftarrow \theta_i - Z_i \cdot se_i$
    $T1 \leftarrow augment\left(M^T, Z\right)$
    $T2 \leftarrow augment\left(T1, \theta\right)$
    $T3 \leftarrow augment(T2, se)$
    $T \leftarrow augment(T3, bound)$
    $S \leftarrow csort\left(T, 3\right)$
    $S$
```

Function enoughruns calls function bootstrap_mean n times to obtain a vector M of bootstrapped means. It then sorts the elements of M in ascending order and finds the (α/2) percentile and (1-α/2) percentile elements of M. If both these %ile elements are within the interval specified by function interval above, (+/- 10% of the overall median) it returns the value enoughruns = 1, otherwise 0.

```
$enoughruns\left(n, X, \alpha, \gamma\right) :=$
    for $i \in 1..n$
        $Xb \leftarrow bootstrap\_k(X)$
        $M_i \leftarrow mean(Xb)$
    $med \leftarrow median(M)$
    $perc\_intvl \leftarrow perc\_interval\left(M, \alpha\right)$
    $boundvec \leftarrow interval\left(med, \gamma\right)$
    $enough_1 \leftarrow 1$  if  $boundvec_1 < perc\_intvl_1$
    $enough_1 \leftarrow 0$  otherwise
    $enough_2 \leftarrow 1$  if  $perc\_intvl_2 < boundvec_2$
    $enough_2 \leftarrow 0$  otherwise
    $\left(stop \leftarrow 1\right)$  if  $enough = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
    $stop \leftarrow 0$  otherwise
    $shape \leftarrow \frac{perc\_intvl_2 - med}{med - perc\_intvl_1}$
    $\begin{bmatrix} stop \\ boundvec_1 \\ perc\_intvl_1 \\ med \\ perc\_intvl_2 \\ boundvec_2 \\ shape \end{bmatrix}$
```

Function rep_enoughruns replicates function enoughruns n times.

```
rep_enoughruns(reps, n, X, α, γ) :=
    for i ∈ 1 .. reps
        $ER_i$ ← $enoughruns(n, X, α, γ)_1$
    m ← mean(ER)
    m
```

Function whatis_enough determines how many runs are enough using Candidate S2

```
whatis_enough(n, X, α, γ, step, start) :=
    r ← rows(X)
    for i ∈ start .. r
        ER ← 0
        if mod(i, step)=0
            Xi ← submatrix(X, 1, i, 1, 2)
            Xiw ← weightPks(Xi)
            ER ← enoughruns(n, Xiw, α, γ)
        if $ER_1$=1
            runs ← i
            mean ← $ER_4$
            break
    $\begin{pmatrix} runs \\ mean \end{pmatrix}$
```

Function rep_enoughruns replicates function whatis_enough n times.

```
rep_whatis_enough(reps, n, X, α, γ, step, start) :=
    for i ∈ 1 .. reps
        $ER^{<i>}$ ← whatis_enough(n, X, α, γ, step, start)
    $ER^T$
```

Function pred_std takes a vector X and its counterpart Xw that is equally weighted according to Firings, the significance level α and the relative error γ and returns a prediction of the number of runs to go using Candidate P2, using the bootstrap estimate of standard error

```
pred_boot(X, Xw, α, γadj, se) :=
    n ← rows(Xw)
    avgfirings ← mean($X^{<2>}$)
    θ ← mean(Xw)
    for j ∈ 1 .. 500
        n_new ← n + floor(j·avgfirings)
        se_new ← $\frac{se \cdot \sqrt{n}}{\sqrt{n\_new}}$
        tvalue ← $qt\left(1 - \frac{\alpha}{2}, n\_new - 1\right)$
        hl_new ← tvalue·se_new
        ratio_new ← $\frac{hl\_new}{|\theta|}$
        runstogo ← j
        break if ratio_new<γadj
    runstogo
```

Function bootstrap_se takes a vector X and the number of bootstrap replications n and returns the bootstrap estimate of standard error

```
bootstrap_se(n,X) := | for $i \in 1..n$
                     |     $\theta b_i \leftarrow \mathrm{bootmean}(X)$
                     | $sdb \leftarrow \mathrm{stdev}(\theta b)$
                     | $boot\_se \leftarrow sdb$
                     | boot_se
```

Function pred_std takes a vector X and its counterpart Xw that is equally weighted according to Firings, the significance level α and the relative error γ and returns a prediction of the number of runs to go using Candidate P1

```
pred_std(X,Xw,α,γ) := | $n \leftarrow \mathrm{rows}(Xw)$
                      | $avgfirings \leftarrow \mathrm{mean}\left(X^{<2>}\right)$
                      | $\theta \leftarrow \mathrm{mean}(Xw)$
                      | $sd \leftarrow \sqrt{\mathrm{var}(Xw)} \cdot \sqrt{\frac{n}{n-1}}$
                      | $hl \leftarrow \mathrm{CIhalflength}(Xw,\alpha)$
                      | $ratio \leftarrow \frac{hl}{|\theta|}$
                      | $\gamma adj \leftarrow \frac{\gamma}{1+\gamma}$
                      | for $j \in 1..500$
                      |     | $n\_new \leftarrow n + \mathrm{floor}(j \cdot avgfirings)$
                      |     | $se\_new \leftarrow \frac{sd}{\sqrt{n\_new}}$
                      |     | $tvalue \leftarrow \mathrm{qt}\left(1 - \frac{\alpha}{2}, n\_new - 1\right)$
                      |     | $hl\_new \leftarrow tvalue \cdot se\_new$
                      |     | $ratio\_new \leftarrow \frac{hl\_new}{|\theta|}$
                      |     | $runstogo \leftarrow j$
                      |     | break if $ratio\_new < \gamma adj$
                      | $\begin{bmatrix} runstogo \\ avgfirings \\ \theta \\ n \\ ratio \\ n\_new \\ ratio\_new \end{bmatrix}$
```

Function stopcrit takes a vector X, the significance level α and the relative error γ and determines when enough runs have been made using Candidate S1.

```
$stopcrit(X, \alpha, \gamma) :=$
    $q \leftarrow rows(X)$
    $Xs \leftarrow csort(X, 1)$
    $max \leftarrow Xs_{q,1}$
    $z_{max+1} \leftarrow 0$
    $\gamma adj \leftarrow \frac{\gamma}{1+\gamma}$
    for $j \in 20..q$
        $Xj \leftarrow submatrix(X, 1, j, 1, 2)$
        $Xjw \leftarrow weightPks(Xj)$
        $n \leftarrow rows(Xjw)$
        $se_j \leftarrow \frac{stdev(Xjw) \cdot \sqrt{\frac{n}{n-1}}}{\sqrt{n}}$ if $n > 1$
        $se_j \leftarrow 2^{1020}$ otherwise
        $\theta_j \leftarrow mean(Xjw)$
        $hl \leftarrow qt\left(1 - \frac{\alpha}{2}, n-1\right) \cdot se_j$
        $CILow_j \leftarrow \theta_j - hl$
        $CIUp_j \leftarrow \theta_j + hl$
        $ratio_j \leftarrow \frac{hl}{\theta_j}$
        $stop \leftarrow 1$ if $ratio_j < \gamma adj$
        $stop \leftarrow 0$ otherwise
        $STOP_1 \leftarrow 1$ if $stop = 1$
        $STOP_1 \leftarrow 0$ otherwise
        $STOP_2 \leftarrow 1$ if $j > 20$
        $STOP_2 \leftarrow 0$ otherwise
        if $STOP = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
            $pred_j \leftarrow 0$
            break
        $pred_j \leftarrow pred\_std(Xj, Xjw, \alpha, \gamma adj)_1$ otherwise
    $\begin{bmatrix} j \\ CILow_j \\ \theta_j \\ CIUp_j \\ ratio_j \\ pred \end{bmatrix}$
```

Function stopboot determines when enough runs have been made using the bootstrap estimate of standard error and calls function pred_boot after each run to predict the number of runs to go using the bootstrap estimate of standard error

```
$\text{stopboot}(X, b, \alpha, \gamma, \text{pilotb}) :=$
    $q \leftarrow \text{rows}(X)$
    $\gamma\text{adj} \leftarrow \frac{\gamma}{1+\gamma}$
    for $j \in \text{pilotb}..q$
        $Xj \leftarrow \text{submatrix}(X, 1, j, 1, 2)$
        $Xjw \leftarrow \text{weightPks}(Xj)$
        $n \leftarrow \text{rows}(Xjw)$
        $se_j \leftarrow \text{bootstrap\_se}(b, Xjw)$  if  $n>1$
        $se_j \leftarrow 2^{1020}$  otherwise
        $\theta_j \leftarrow \text{mean}(Xjw)$
        $hl \leftarrow qt\left(1 - \frac{\alpha}{2}, n-1\right) \cdot se_j$
        $CILow_j \leftarrow \theta_j - hl$
        $CIUp_j \leftarrow \theta_j + hl$
        $ratio_j \leftarrow \frac{hl}{\theta_j}$
        $stop \leftarrow 1$  if  $ratio_j < \gamma\text{adj}$
        $stop \leftarrow 0$  otherwise
        $STOP_1 \leftarrow 1$  if  $stop = 1$
        $STOP_1 \leftarrow 0$  otherwise
        $STOP_2 \leftarrow 1$  if  $j > 20$
        $STOP_2 \leftarrow 0$  otherwise
        if  $STOP = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
            $pred_j \leftarrow 0$
            break
        $pred_j \leftarrow \text{pred\_boot}(Xj, Xjw, \alpha, \gamma\text{adj}, se_j)$  otherwise
    $\begin{bmatrix} j \\ CILow_j \\ \theta_j \\ CIUp_j \\ ratio_j \\ pred \end{bmatrix}$
```

ANIMATION/GRAPHING FUNCTIONS:

Function animate_bootstrap_1 replicates the bootstrap resampling (1 pick at a time) of a vector X in order to show 1 rep of the resampling process in the animated graphs below

```
animate_bootstrap_1(X,p) :=
    Xp ← submatrix(X, 1, p, 1, 1)
    p ← rows(Xp)
    Xs ← sort(Xp)
    max ← $Xs_p$
    $z_{max+1} \leftarrow 0$
    for i ∈ 1 .. p
        for j ∈ 1 .. p
            $A_j \leftarrow$ bootstrap_1(X)
        $gen_i \leftarrow A_j$
        $z_{(A_j+1)} \leftarrow z_{(A_j+1)} + 1$
        $Z_i \leftarrow z$
        $I_i \leftarrow i$
    $\begin{pmatrix} gen \\ Z \\ I \end{pmatrix}$
```

Function animate_bootstrap_k replicates the bootstrap resampling process (k picks at a time) of a vector X in order to show n reps of the resampling process in the animated graphs below. α is the significance level, γ is the relative error, and p is the number of data points to consider.

```
animate_bootstrap_k(n, X, α, γ, p) :=
    $Xp \leftarrow \mathrm{submatrix}(X, 1, p, 1, 1)$
    $Xs \leftarrow \mathrm{sort}(Xp)$
    $max \leftarrow Xs_p$
    $z_{max+1} \leftarrow 0$
    $y_{max+1} \leftarrow 0$
    $\gamma adj_1 \leftarrow \frac{\gamma}{1+\gamma}$
    $\Sigma 1 \leftarrow 0$
    $\Sigma 2 \leftarrow 0$
    for $i \in 1..n$
        for $j \in 1..p$
            $A_j \leftarrow \mathrm{bootstrap\_1}(Xp)$
            $y_{(A_j+1)} \leftarrow y_{(A_j+1)} + 1$
        $As \leftarrow \mathrm{sort}(A)$
        $\theta b_i \leftarrow \mathrm{floor}(\mathrm{mean}(A))$
        $\theta_i \leftarrow \mathrm{floor}(\mathrm{mean}(\theta b))$
        $\gamma adj_i \leftarrow \frac{\gamma}{1+\gamma}$
        $int \leftarrow \mathrm{interval}(\theta, \gamma)$
        $\Sigma 1 \leftarrow \Sigma 1 + \theta b_i$
        if $i > 1$
            $se\theta_i \leftarrow \left[ \frac{\sum_{m=1}^{i} \left| \theta b_m - \frac{\Sigma 1}{i} \right|}{(i-1)} \right]^{\frac{1}{2}}$
            $ratio_i \leftarrow \frac{qt(1-\alpha, i) \cdot se\theta_i}{\theta_i}$
        otherwise
            $se\theta_i \leftarrow 0$
            $ratio_i \leftarrow 0$
        $gen_i \leftarrow A$
        $YY_i \leftarrow y$
        $z_{(\theta b_i)} \leftarrow z_{\theta b_i} + 1$
        $Z_i \leftarrow z$
        $\Theta_i \leftarrow \mathrm{mean}(\theta)$
        $SE\Theta_i \leftarrow se\theta$
        $RATIO_i \leftarrow ratio$
        $\Gamma_i \leftarrow \gamma adj$
    $\Theta B_i \leftarrow \theta b$
  $\begin{bmatrix} \mathrm{gen} \\ Z \\ YY \\ \mathrm{se}\theta \\ SE\Theta \\ \theta \\ \Theta \\ \mathrm{ratio} \\ RATIO \\ \gamma \mathrm{adj} \\ \Gamma \\ \theta b \\ \Theta B \end{bmatrix}$
```

Function range determines the range of data in the subvector of size p of the vector X of size k so the range of the animated graphs can be automatically set

```
$\mathrm{range}(X,p) :=$
  $r \leftarrow p$
  $Xp \leftarrow \mathrm{submatrix}(X,1,p,1,1)$
  $Xs \leftarrow \mathrm{sort}(Xp)$
  $\mathrm{max} \leftarrow Xs_r$
  $\mathrm{min} \leftarrow Xs_1$
  $\begin{pmatrix} \mathrm{min} \\ \mathrm{max} \end{pmatrix}$
```

Function animate_stopcrit animates the process of function stopcrit.

```
$anim\_stopcrit(X,\alpha,\gamma) :=$
    $q \leftarrow rows(X)$
    $Xs \leftarrow csort(X,1)$
    $max \leftarrow Xs_{q,1}$
    $z_{max+1} \leftarrow 0$
    for $j \in 1..q$
        $Xj \leftarrow submatrix(X,1,j,1,2)$
        $Xqw \leftarrow weightPks(Xj)$
        $n \leftarrow rows(Xqw)$
        $se_j \leftarrow \frac{stdev(Xqw)\cdot\sqrt{\frac{n}{n-1}}}{\sqrt{n}}$ if $n>1$
        $\theta_j \leftarrow mean(Xqw)$
        $ratio_j \leftarrow \frac{qt\left(1-\frac{\alpha}{2},n-1\right)\cdot se_j}{\theta_j}$
        $CIlow_j \leftarrow \theta_j - qt\left(1-\frac{\alpha}{2},n-1\right)\cdot se_j$
        $CIhi_j \leftarrow \theta_j + qt\left(1-\frac{\alpha}{2},n-1\right)\cdot se_j$
        $\gamma adj_j \leftarrow \frac{\gamma}{1+\gamma}$
        $stop \leftarrow 1$ if $ratio_j < \gamma adj_j$
        $stop \leftarrow 0$ otherwise
        $\Theta_j \leftarrow \theta$
        $SE_j \leftarrow se$
        $RATIO_j \leftarrow ratio$
        $\Gamma adj_j \leftarrow \gamma adj$
        $XX_j \leftarrow X^{<1>}$
        $z_{\left(X_{j,1}+1\right)} \leftarrow z_{\left(X_{j,1}+1\right)} + 1$
        $Z_j \leftarrow z$
    $\begin{bmatrix} se \\ SE \\ \theta \\ \Theta \\ ratio \\ RATIO \\ \gamma adj \\ \Gamma adj \\ z \\ Z \\ CIlow \\ CIhi \end{bmatrix}$
```

DATA FOR
ANIMATED BOOTSTRAP SAMPLING
PROGRAM using Mathcad 6.0

(Based on RAND Corp.'s *enoughruns.F* Fortran program designed to determine when enough BRAWLER runs have been made)

0.4	5	0	2
0.33	3	0.33	3
0.5	4	0.5	2
0.5	4	0.2	5
0.2	5	0	1
0.25	4	0.67	3
1	1	0	4
0.4	5	0	4
0.5	2	0	3
1	1	0.4	5
0	1	0	0
0.5	2	0.33	3
0.5	2	0	2
0	1	0.5	2
0	1	0	2
0	0	0	0
0	2	0.25	4
0.33	3	0.33	3
0.5	2	1	1
0	0	0.5	2
0.67	3	0	2
0	0	0.5	4
1	2	0	3
0	0	0.14	7
1	2	0	0
0	4	1	1
0.5	4	0.5	2
0	1	0.5	4
0.33	3	1	2
1	1	0.67	3
0	3	0.5	2
0	7	1	2
1	2	0	5
0	1	0.14	7
0	0	0	3
0.17	6	0.33	3
0	1	0.33	3
0.67	3	0	2
0.2	5	0	1
0.5	4	0	0

GLOBAL PARAMETERS:

$B := 501$

$p := 99$

$q := 122$

$\alpha := .10$

$\gamma := .10$

$$Pk1 := \begin{pmatrix} 0.33 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0.25 \\ 0 \\ 0.5 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0.25 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0.5 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0.33 \\ 1 \\ 1 \\ 1 \\ 0.17 \\ 0.33 \\ 0.14 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0.5 \\ 0.25 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad F1 := \begin{pmatrix} 3 \\ 1 \\ 0 \\ 2 \\ 0 \\ 2 \\ 4 \\ 4 \\ 2 \\ 1 \\ 2 \\ 0 \\ 1 \\ 4 \\ 3 \\ 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 1 \\ 3 \\ 2 \\ 1 \\ 1 \\ 6 \\ 3 \\ 7 \\ 1 \\ 1 \\ 0 \\ 0 \\ 4 \\ 6 \\ 4 \\ 5 \\ 1 \\ 2 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

Y is the output data Adjusted
Kills per Firing and Firings from BRAWLER

$Data1 := (stack(Pk1, Pk2)) \cdot 100$

$Data2 := stack(F1, F2)$

$Y := augment(Data1, Data2)$

Y is the BRAWLER data matrix
of AKPF's and Firings.
Y must be made of integers
only, therefore the AKPF's
are multiplied by 100
because Mathcad
animated simulations only
work with integer data

$$Y = \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 40 & 5 \\ 2 & 33 & 3 \\ 3 & 50 & 4 \\ 4 & 50 & 4 \\ 5 & 20 & 5 \\ 6 & 25 & 4 \\ 7 & 100 & 1 \\ 8 & 40 & 5 \\ 9 & 50 & 2 \\ 10 & 100 & 1 \\ 11 & 0 & 1 \\ 12 & 50 & 2 \\ 13 & 50 & 2 \\ 14 & 0 & 1 \\ 15 & 0 & 1 \end{array}$$

$k := rows(Y)$

$k = 200$

k is the # of
data points (runs)

$$Pk2 := \begin{pmatrix} 0 \\ 0.5 \\ 0 \\ 0.75 \\ 0.17 \\ 0.33 \\ 0.5 \\ 0.25 \\ 0 \\ 0 \\ 0.33 \\ 0.33 \\ 0 \\ 0.25 \\ 0.33 \\ 0 \\ 0 \\ 0.5 \\ 1 \\ 0.67 \\ 0.4 \\ 0.25 \\ 0 \\ 0 \\ 0.5 \\ 0.75 \\ 0.33 \\ 0.5 \\ 0 \\ 1 \\ 0 \\ 0.2 \\ 0.33 \\ 0 \\ 0 \\ 0.33 \\ 0.5 \\ 0.43 \\ 0.2 \\ 1 \\ 0 \\ 0.33 \\ 0 \\ 0 \end{pmatrix} \qquad F2 := \begin{pmatrix} 1 \\ 2 \\ 1 \\ 4 \\ 6 \\ 3 \\ 2 \\ 4 \\ 1 \\ 0 \\ 3 \\ 3 \\ 0 \\ 4 \\ 3 \\ 2 \\ 3 \\ 2 \\ 2 \\ 3 \\ 5 \\ 8 \\ 1 \\ 4 \\ 2 \\ 4 \\ 6 \\ 4 \\ 0 \\ 1 \\ 2 \\ 5 \\ 3 \\ 1 \\ 1 \\ 3 \\ 2 \\ 7 \\ 5 \\ 2 \\ 3 \\ 3 \\ 2 \\ 3 \end{pmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0.33 \\ 0.17 \\ 0 \\ 0 \\ 0.33 \\ 0.67 \\ 0 \\ 0 \\ 0.5 \\ 0 \\ 1 \\ 0.5 \\ 0.33 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 1 \\ 3 \\ 6 \\ 0 \\ 2 \\ 3 \\ 3 \\ 0 \\ 0 \\ 2 \\ 4 \\ 2 \\ 4 \\ 6 \\ 1 \end{bmatrix}$$

$Yp := \text{submatrix}(Y, 1, p, 1, 2)$

$Yq := \text{submatrix}(Y, 1, q, 1, 2)$

$Ys := \text{csort}(Yp, 1)$

$$\begin{bmatrix} 0.1 \\ 0 \\ 0.4 \\ 1 \\ 0.33 \\ 0.33 \\ 0 \\ 1 \\ 0.5 \\ 0.33 \\ 0 \\ 0.5 \\ 0 \\ 1 \\ 0 \\ 0.5 \end{bmatrix} \quad \begin{bmatrix} 10 \\ 1 \\ 5 \\ 1 \\ 3 \\ 3 \\ 0 \\ 1 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$Yw := \text{weightPks}(Y)$

$\text{sumfirings}(Y) = 506$

$\text{mean}(Yw) = 32.178$

Yw =

	1
1	40
2	40
3	40
4	40
5	40
6	33
7	33
8	33
9	50
10	50
11	50
12	50
13	50
14	50
15	50

$Ypw := \text{weightPks}(Yp)$

$\text{sumfirings}(Yp) = 229$

$\text{mean}(Ypw) = 34.048$

Ypw =

	1
1	40
2	40
3	40
4	40
5	40
6	33
7	33
8	33
9	50
10	50
11	50
12	50
13	50
14	50
15	50

$Yqw := \text{weightPks}(Yq)$

$\text{sumfirings}(Yq) = 287$

$\text{mean}(Yqw) = 32.387$

Yqw =

	1
1	40
2	40
3	40
4	40
5	40
6	33
7	33
8	33
9	50
10	50
11	50
12	50
13	50
14	50
15	50

$\mathrm{CIvec1} := \mathrm{boot\_CIs}(Y, B, \alpha, 1)$

$\mathrm{meanvec1} := \mathrm{CIvec1}_1$

meanvec1 =

	1
1	40
2	37.375
3	41.583
4	43.688
5	38.19
6	35.8
7	37.923
8	38.484
9	39.212
10	41.088
11	39.771
12	40.622
13	40.897
14	40.05
15	39.171

$\mathrm{CIlowvec1} := \mathrm{CIvec1}_2$

CIlowvec1 =

	1
1	40
2	35.625
3	38.5
4	40.75
5	33.762
6	32.16
7	33.269
8	34.484
9	35.061
10	36.5
11	35.057
12	35.459
13	36.769
14	35.25
15	34.024

$\mathrm{CIupvec1} := \mathrm{CIvec1}_3$

CIupvec1 =

	1
1	40
2	39.125
3	44.917
4	46.438
5	42.19
6	40.04
7	43.808
8	43.258
9	44.03
10	46.853
11	45.257
12	45.432
13	45.718
14	45.65
15	44.171

boot_Z(Ypw, B) =

	1	2	3	4	5	6
1	28.62	1.941	-2.797	34.048	2.072	39.843
2	28.803	1.939	-2.705	34.048	2.072	39.653
3	29.013	1.981	-2.542	34.048	2.072	39.314
4	28.983	2	-2.533	34.048	2.072	39.297
5	29.218	1.938	-2.492	34.048	2.072	39.212
6	29.437	1.903	-2.424	34.048	2.072	39.07
7	29.38	1.988	-2.348	34.048	2.072	38.913
8	29.738	1.841	-2.341	34.048	2.072	38.899
9	29.886	1.806	-2.304	34.048	2.072	38.823
10	30.109	1.751	-2.25	34.048	2.072	38.71
11	29.694	1.941	-2.244	34.048	2.072	38.697
12	29.703	1.981	-2.193	34.048	2.072	38.593
13	30	1.848	-2.19	34.048	2.072	38.586
14	29.738	1.976	-2.182	34.048	2.072	38.569
15	29.913	1.929	-2.144	34.048	2.072	38.49

$$\mathrm{enoughruns}(B, \mathrm{Ypw}, \alpha, \gamma) = \begin{bmatrix} 0 \\ 30.628 \\ 30.843 \\ 34.031 \\ 37.817 \\ 37.434 \\ 1.188 \end{bmatrix}$$

$\text{rep\_enoughruns}(1, B, Ypw, \alpha, \gamma) = 1$

$$\text{whatis\_enough}(B, Y, \alpha, \gamma, 1, 180) = \begin{pmatrix} 180 \\ 32.639 \end{pmatrix}$$

$\text{rep\_whatis\_enough}(1, B, Y, \alpha, \gamma, 1, 180) = (\ 180 \quad 32.704\ )$

$$\text{stopcrit}(Y, \alpha, \gamma) = \begin{bmatrix} 122 \\ 29.448184 \\ 32.38676 \\ 35.325335 \\ 0.090734 \\ \{-768170609, 1073080493\} \end{bmatrix}$$

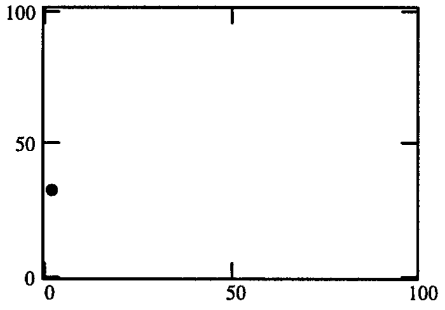
$res := \text{animate\_bootstrap\_1}(Yw, p)$

$r := Ys_{1,1} + 1 .. Ys_{p,1} + 1$

$gen := res_1 \quad Z := res_2$

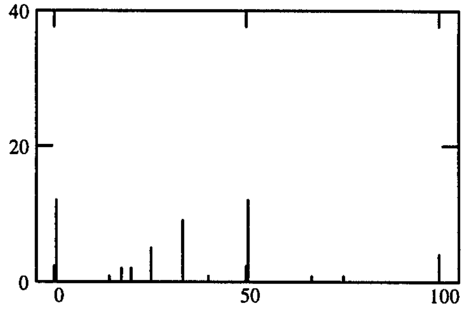
$I := res_3$

Animated Bootstrap Resampling choose k out of k with replacement (1 frame = 1 choose)



$gen_{FRAME+1}, r$ = values of bootstrapped data points

Animated Bootstrap Resampling Histogram (Cumulative) choose k out of k with replacement (1 frame = 1 choose)



r = value of bootstrapped data point

$j := 1 .. p$ $\qquad$ $res := \text{animate\_bootstrap\_k}(B, Y, \alpha, \gamma, p)$

$b := 1 .. B$ $\qquad$ $gen := res_1$ $\quad$ $Z := res_2$ $\quad$ $YY := res_3$ $\quad$ $se\theta := res_4$ $\quad$ $SE\Theta := res_5$ $\quad$ $\theta := res_6$

$\gamma\gamma := res_{10}$ $\quad$ $\Gamma := res_{11}$ $\quad$ $\theta b := res_{12}$ $\quad$ $\Theta B := res_{13}$ $\quad$ $\Theta := res_7$ $\quad$ $ratio := res_8$

$RATIO := res_9$

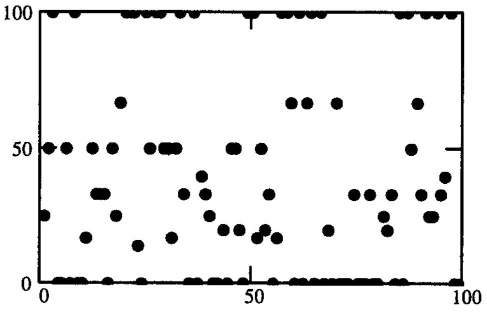
```
range(X,p) :=  | r←p
               | Xp←submatrix(X,1,p,1,1)
               | Xs←sort(Xp)
               | max←Xs_r
               | min←Xs_1
               | (min)
               | (max)
```

$$\text{range}(Y,p) = \begin{pmatrix} 0 \\ 100 \end{pmatrix}$$

$r := \text{range}(Y,p)_1 + 1 .. \text{range}(Y,p)_2 + 1$

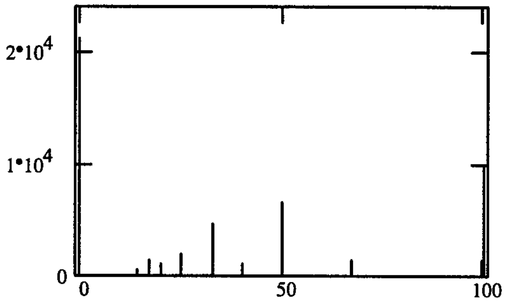
Animated Resampling

B bootstrap samples (1 frame = 1 bootstrap sample for each of B bootstrap reps)



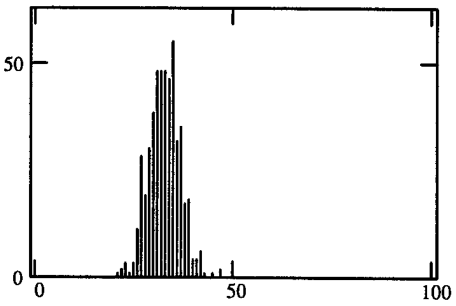
Cumulative Animated Bootstrap Sample Histogram

B bootstrap samples (1 frame = 1 bootstrap sample for each of B bootstrap reps)

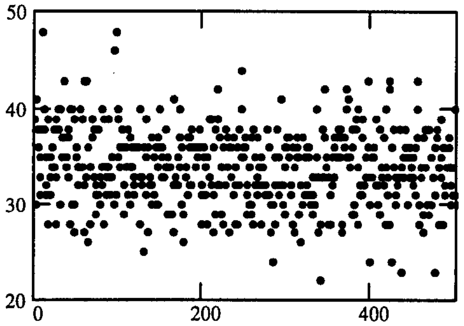


Animated Bootstrap Means (Cumulative)

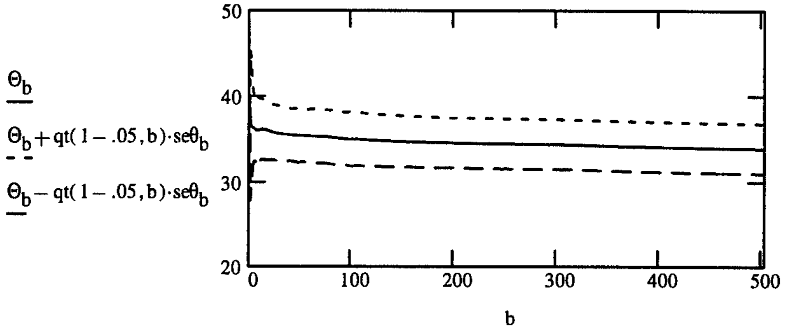
B bootstrap means (1 frame = 1 median for each of B bootstrap reps)



Mean of Boot Rep



Grand Mean Across Boot Reps and 100(1-α)% confidence bounds



$CIlowvec := anim\_stopcrit(Y,\alpha,\gamma)_{11}$

$meanvec := anim\_stopcrit(Y,\alpha,\gamma)_{3}$

$CIupvec := anim\_stopcrit(Y,\alpha,\gamma)_{12}$

meanvec =

	1
1	40
2	37.375
3	41.583
4	43.688
5	38.048
6	35.96
7	38.423
8	38.677
9	39.364
10	41.147
11	39.971
12	40.514
13	41
14	39.975
15	39

CIlowvec =

	1
1	40
2	34.781
3	37.871
4	40.531
5	33.43
6	31.78
7	32.552
8	33.813
9	34.739
10	35.719
11	34.334
12	35.159
13	35.901
14	34.711
15	33.608

CIupvec =

	1
1	40
2	39.969
3	45.295
4	46.844
5	42.665
6	40.14
7	44.294
8	43.542
9	43.989
10	46.575
11	45.608
12	45.868
13	46.099
14	45.239
15	44.392

$j := 1 .. q$

$res := anim\_stopcrit(Y,\alpha,\gamma)$

$r := Ys_{1,1} .. Ys_{p,1}$

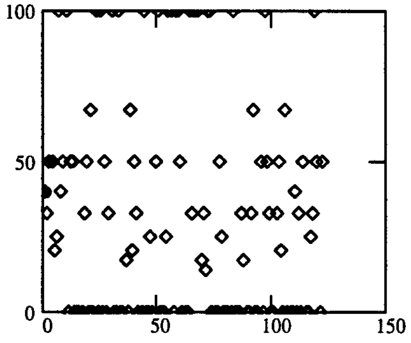
$se := res_1$ $SE := res_2$ $\theta := res_3$ $\Theta := res_4$ $ratio := res_5$ $RATIO := res_6$

$b := 1 .. B$

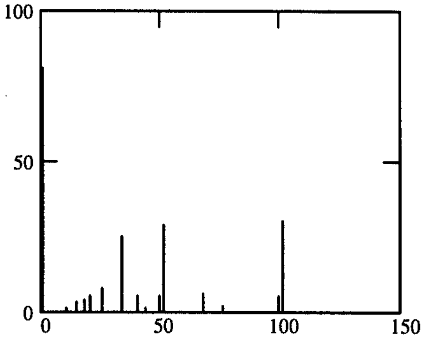
$\gamma adj := res_7$ $\Gamma adj := res_8$ $z := res_9$ $Z := res_{10}$

$r := range(Y,q)_1 + 1 .. range(Y,q)_2 + 1$

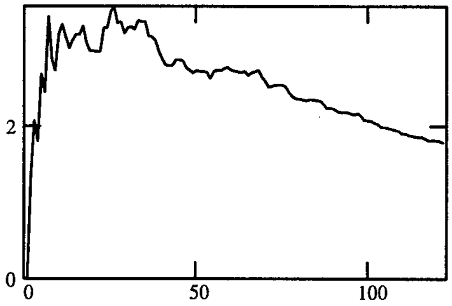
Animated Data from Runs



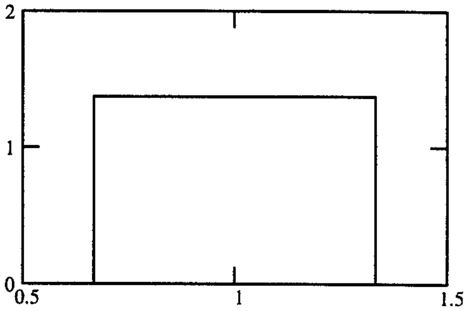
Animated Data from Runs Histogram



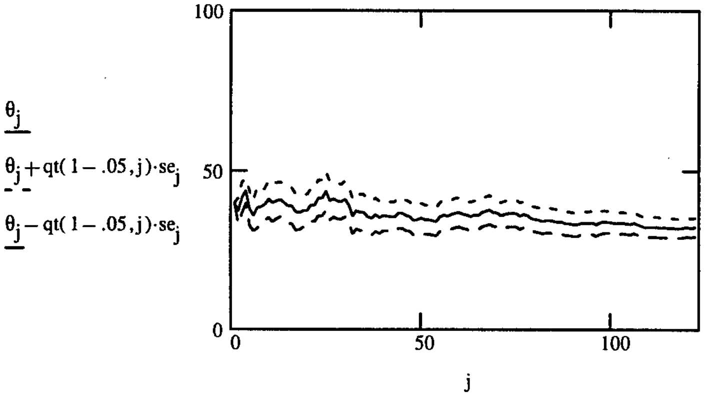
Non-Bootstrap Estimate of Standard Error vs. # Runs



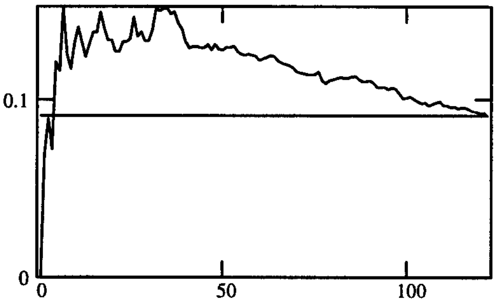
Animated Standard Error



Grand Mean Across Reps and 100(1-α)% confidence bounds



Ratio of Half-Length to Mean and Threshold



Appendix B. *enoughruns.F*

```
      program enoughruns
C*** THIS ROUTINE IS WRITTEN AT RAND
C#ABSTRACT Program to test whether enough tbmain runs have been made
C#PURPOSE  Find the mean and confidence interval for exchange ratio
C#AUTHOR   Carol Johnson (bootstrap method from Tom Lucas, RAND)
C#TYPE     Postprocessing
C#PARAMETER DESCRIPTIONS:
CIN  inarg(1)  CHAR - (optional) 'eratio' or 'red' or 'blue'
C#TECHNICAL DESCRIPTION:
C Program enoughruns tells whether "enough" tbmain runs have been made.
C Used by the fly script.
C Input file: fly.TOTALS, with lines like:
C:0           1 BLUE AC +  1 RED AC =  2 TOTAL.    0 BLUE AC ALIVE + ...
C Output file: fly.PERMEAS
C stdout: "yes, ___ runs are enough" or "no, ___ runs are not enough"
C#VARIABLE DESCRIPTIONS:
C  MAX_RUNS PAR - maximum number of lines in fly.TOTALS to process
C  iargc    INT FUNCTION - number of arguments on the command line
C  nargs    INT - number of arguments on the command line
C  indexpm  INT - 1, 2, or 3 for 3 argument possibilities
C  num_runs INT - number of lines in fly.TOTALS
C  bac      INT - number of blue a/c
C  balive   INT - number of blue a/c alive at the end of this run
C  bkrun    INT array - number of blue a/c killed, in each run
C  rac      INT - number of red a/c
C  ralive   INT - number of red a/c alive at the end of this run
C  rkrun    INT array - number of red a/c killed, in each run
C  pmmean   REAL - performance measure's mean
C  pmmin    REAL - performance measure's least value
C  pmlow    REAL - performance measure's 5th percentile
C  pmmedian REAL - performance measure's median
C  pmhigh   REAL - performance measure's 95th percentile
C  pmmax    REAL - performance measure's greatest value
C  inarg(1) CHAR - argument, or default if no argument
C  line1    CHAR - a line of the fly.TOTALS file, as read in
C  answer   CHAR - 'yes' or 'no', enough runs
C####
C#AUDIT
C BY tacb IN v6.15 tacb.cruise ON 95/06/16 18:09:27 to.n.rou
C BY tacb IN v6.15 tacb.cruise ON 95/06/16 17:57:04 MODIFIED
C  Putting illegal unit -1 into a "variable", to satisfy H-P
C BY tacb IN v6.15 skip.cnj ON 94/12/28 13:53:03 to.n.rou
C BY tacb IN v6.15 skip.cnj ON 94/12/21 13:50:59 MODIFIED
C  Split out subroutine bootstrap, to call from analyze pgm.
C BY tacb IN v6.15 skip.cnj ON 94/12/21 13:47:55 to.n.rou
C BY tacb IN v6.15 skip.cnj ON 94/12/20 13:43:56 MODIFIED
C  Add argument to choose measure of performance
C BY tacb IN v6.15 tacb.rand ON 94/10/17 14:50:31 to.n.rou
C BY tacb IN v6.15 tacb.rand ON 94/10/10 16:45:37 MODIFIED
C  Change to bootstrap method a la Tom Lucas, RAND.
C  Loop to find minimum number of "enough" runs.
C  Add field "#Runs" to EXCHANGE RATIO STATISTICS
C BY tacb IN v6.15 tacb.rand ON 94/09/20 17:01:56 to.n.rou
C BY tacb IN v6.15 tacb.rand ON 94/09/20 14:21:39 MODIFIED
C  First test was of runs 1-11; now is of runs 10-20
```

```
C BY tacb IN v6.15 tacb.analyze ON 94/09/13 11:56:46 to.n.rou
C BY tacb IN v6.15 tacb.analyze ON 94/09/13 11:49:26 MODIFIED
C  Expect every line to be a TOTAL. line
C BY tacb IN v6.15 tacb.analyze ON 94/09/13 11:18:54 to.n.rou
C####
      integer   MAX_RUNS
      parameter(MAX_RUNS=1001)
      integer iargc, nargs,indexpm
      integer num_runs,i,minus1
      integer bac,balive,bkrun(MAX_RUNS)
      integer rac,ralive,rkrun(MAX_RUNS)
      real pmmean,pmmin,pmlow,pmmedian,pmhigh,pmmax
      character*70 inarg(1)
      character*80 line1
      character*3  answer
C*ENDDEC
C     --GET ARGUMENTS
      minus1 = -1
      nargs=iargc()
      if (nargs.eq.0) then
       inarg(1) = 'eratio'
       indexpm = 1
      else
       call getarg(1, inarg(1))
       if (inarg(1).eq.'eratio') then
          indexpm = 1
       elseif (inarg(1).eq.'red') then
          indexpm = 2
       elseif (inarg(1).eq.'blue') then
          indexpm = 3
       else
          write(6,*) 'NOTE analyze gets unknown argument '//inarg(1)
          write(6,*)      '     so writing illegal unit -1 to abort'
          write(minus1,*) '     so writing illegal unit -1 to abort'
       endif
      endif

C     --INITIALIZE FILES AND COUNTS AND SUCH
      open(unit=5,file='fly.TOTALS',status='old',err=2000)
      open(unit=11,file='fly.PERMEAS',status='unknown',err=3000)

C     --READ NUMBERS OF BLUE AND RED AIRCRAFT KILLED IN SEVERAL RUNS
      do 100 num_runs=1,MAX_RUNS
       read(5,fmt='(a)',end=200,err=2000) line1
       if (line1(40:45) .eq. 'TOTAL.' ) then
C           -- Find number of aircraft killed in this run
          read(line1,'( t12,i2,t25,i2,t49,i2,17x,i2)')
     >        bac, rac, balive, ralive
          bkrun(num_runs) = bac - balive
          rkrun(num_runs) = rac - ralive
       else
          goto 2100
       endif
100   continue
200   continue
      num_runs=num_runs -1

C     --BOOTSTRAP, IF A MINIMUM NUMBER OF RUNS
      if (num_runs.ge.20) then
       call bootstrap('enoughruns',indexpm,11,num_runs,bkrun,rkrun,20,
     >    pmmean,pmmin,pmlow,pmmedian,pmhigh,pmmax,answer)
```

```
      else
       answer = 'no'
       write(11,*) ' '
       write(11,fmt='(''no, '',i3,'' runs are not enough'')')
     >    num_runs
       write(6,fmt='(''no, '',i3,'' runs are not enough'')')
     >    num_runs
      endif
      stop

2000  write (6,2001)
2001  format     ('enoughruns: File fly.TOTALS is not readable')
      goto 9000
2100  write (6,2101) line1
2101  format     ('enoughruns: Expected TOTAL line but found'/a)
      goto 9000
3000  write (6,3001)
3001  format     ('enoughruns: File fly.PERMEAS is not writable')
9000  write(6, *)     '            so writing illegal unit -1 to abort'
      write(minus1,*) '            so writing illegal unit -1 to abort'
      end
```

```
C-----------------------------------------------------------------------
      subroutine bootstrap(caller,ipm,iunit,num_runs,bkrun,rkrun,
     > min_run, pmmean,pmmin,pmlow,pmmedian,pmhigh,pmmax,answer)
C*** THIS SUBROUTINE IS WRITTEN AT RAND
C#ABSTRACT Get statistics on exchange ratio or other performance measure
C#PURPOSE  Help determine whether enough runs have been made
C#AUTHOR   Carol N. Johnson (bootstrap method from Tom Lucas, RAND)
C#TYPE     Post-processing
C#PARAMETER DESCRIPTIONS:
CIN  caller   CHAR - name of the calling program (analyze or enoughruns)
CIN  ipm      INT - performance measure option:
C                   1 = eratio;
C                   2 = red a/c killed;
C                   3 = blue a/c killed
CIN  iunit    INT - Fortran logical unit at which to write
CIN  num_runs INT - maximum number of runs to bootstrap
CIN  bkrun(num_runs) INT - number of blue a/c killed, in each run
CIN  rkrun(num_runs) INT - number of red  a/c killed, in each run
CIN  min_run  INT - minimum number of runs to bootstrap
COUT pmmean   REAL - performance measure's mean
COUT pmmin    REAL - performance measure's least value
COUT pmlow    REAL - performance measure's 5th percentile
COUT pmmedian REAL - performance measure's median
COUT pmhigh   REAL - performance measure's 95th percentile
COUT pmmax    REAL - performance measure's greatest value
COUT answer   CHAR - 'yes' or 'no', enough runs
C#TECHNICAL DESCRIPTION:
C  This is a poor man's way of making N runs 501 times:
C  Given numbers of red and blue aircraft killed in each of N runs:
C  do 501 (or 2001) times:
C     randomly draw N of N runs
C     calculate exchange ratio (or use other measure of performance)
C  sort 501 exchange ratios
C  eratio(251) is the median
C  90% of the 501 eratios are between eratio(26) and eratio(476)
C  when 90% of eratios are within the median plus or minus 10%,
C   N runs are enough
C#VARIABLE DESCRIPTIONS:
C  ITERATIONS   PAR - number of times to simulate making num_runs runs
C  iseq         INT - random number seed
C  i            INT - index
C  now_runs     INT - index for runs
C  ib           INT - index for bootstrap iterations
C  index        INT - randomly-chosen run
C  bksum        INT - sum of blue a/c killed in num_runs random runs
C  rksum        INT - sum of red  a/c killed in num_runs random runs
C  pmsum        REAL - sum of performance measure in num_runs runs
C  permeas      REAL - performance measure in each bootstrap iteration
C  anti         LOGICAL - .true. or .false., antithetical random numbers
C####
C#AUDIT
C BY tacb IN v6.15 tacb.analyze ON 95/10/25 18:42:21 to.n.rou
C BY tacb IN v6.15 tacb.analyze ON 95/10/25 17:25:41 MODIFIED
C  Avoid NaN when median is zero.
C BY tacb IN v6.15 tacb.analyze ON 95/09/18 10:19:11 to.n.rou
C BY tacb IN v6.15 tacb.analyze ON 95/09/15 14:06:57 MODIFIED
C  For analyze program and ipm=6, print combined report for ipm=1+2+3
C BY tacb IN v6.15 tacb.analyze ON 95/08/02 13:47:25 to.n.rou
C BY tacb IN v6.15 tacb.analyze ON 95/08/02 11:00:27 MODIFIED
C  Print which kind of performance measure is being used, in analyze pgm
C  Change indexpm to ipm
```

```
C BY tacb IN v6.15 skip.cnj ON 95/01/31 16:40:32 to.n.rou
C BY tacb IN v6.15 skip.cnj ON 95/01/31 16:31:14 MODIFIED
C  If caller is analyze, say no, not enough if fewer than 20 runs
C BY tacb IN v6.15 skip.cnj ON 94/12/28 14:09:50 to.n.rou
C####
C Args:
      character*(*) caller
      integer ipm,iunit
      integer num_runs, bkrun(num_runs), rkrun(num_runs), min_run
      real pmmean,pmmin,pmlow,pmmedian,pmhigh,pmmax
      character*(*) answer
C Functions:
      real rnsq
C Locals:
      integer   ITERATIONS
      parameter(ITERATIONS= 501)
CUT   parameter(ITERATIONS=2001)
      integer iseq
      integer i,now_runs,ib,index
      integer bksum ,rksum
      real pmsum
      real permeas(ITERATIONS)
      logical anti
C*ENDDEC
       iseq = 0
       anti = .false.
       call ranini(iseq,anti)
       do 700 now_runs=num_runs,min_run,-1
C           --ITERATE, CALCULATING PERFORMANCE MEASURE
          pmsum = 0.
          do 500 ib=1,ITERATIONS
             bksum=0
             rksum=0
             do 220 i = 1, now_runs
              index = NINT(rnsq(iseq)*real(now_runs)+.500001)
              bksum=bksum +bkrun(index)
              rksum=rksum +rkrun(index)
220            continue

             goto (310,320,330), ipm
310            continue
              if (bksum.ne.0.) then
                 permeas(ib) = float(rksum)/float(bksum)
              else
                 permeas(ib) = 1.E+20
              endif
             goto 390
320            continue
              permeas(ib) = float(rksum)/float(now_runs)
             goto 390
330            continue
              permeas(ib) = float(bksum)/float(now_runs)
             goto 390
390            continue

             pmsum = pmsum +permeas(ib)
500         continue
          pmmean = pmsum/real(ITERATIONS)

C           --SORT PERFORMANCE MEASURES ASCENDING, IN PLACE
          call sort(permeas,ITERATIONS)
          pmmin = permeas(1)
          pmmedian = permeas((ITERATIONS+1)/2)
          pmlow = permeas(ITERATIONS/20)
          pmhigh= permeas(ITERATIONS -ITERATIONS/20)
          pmmax = permeas(ITERATIONS)

C           --DECIDE WHETHER ENOUGH RUNS
          if (caller .eq. 'analyze'
     >      .and. now_runs .lt. 20) then
             answer = 'no'
          elseif (pmlow .ge. pmmedian*.90
     >       .and.pmhigh .le. pmmedian*1.10) then
             answer = 'yes'
          else
             answer = 'no'
          endif
          if (caller .eq. 'analyze') then
             call writesix (ipm,iunit,now_runs,
     >          pmmean,pmlow,pmmedian,pmhigh,permeas,ITERATIONS,answer)
          else
             call writenow (ipm,iunit,now_runs,
     >          pmmean,pmlow,pmmedian,pmhigh,permeas,ITERATIONS,answer)
          endif
          if (answer .eq. 'no') goto 800
700      continue
800      continue
       return
      end
```

```
C-----------------------------------------------------------------------
      subroutine writenow (ipm,iunit,now_runs,
     > pmmean,pmlow,pmmedian,pmhigh,permeas,ITERATIONS,answer)
C#PARAMETERS
      integer ipm,iunit,now_runs,ITERATIONS
      real pmmean,pmlow,pmmedian,pmhigh,permeas(ITERATIONS)
      character*(*) answer
C#LOCAL
      real FRACTINC,FRACTMAX1
C     -- FRACTINC, FRACTMAX1 depend on ITERATIONS
      parameter(FRACTINC=.01, FRACTMAX1=.905)
CUT   parameter(FRACTINC=.005,FRACTMAX1=.9025)
      integer i,ib,intervals,tvalpop,sumpop,j
      real fractmin,fractmax,tvalmax
      character*14 pmname(3)
      data pmname/'EXCHANGE RATIO',
     >            'REDS KILLED',
     >            'BLUES KILLED'/
C#ENDDEC
C     --WRITE CONFIDENCE INTERVAL AND MEDIAN OF PERFORMANCE MEASURES
4010  format('        BOOTSTRAP STATISTICS -- ',a//
     >       '                        <---Extrema of 90%--->        '/
     >       '                          5th             95th        '/
     >       ' #Runs   Mean     Min     %ile  Median    %ile     Max'/
     >       ' -----  ------  ------  ------  ------  ------  ------'/
     >       i6,6f8.2,'   Value')
4011  format(14x,5f8.3,'   Fraction of Median')

      write(iunit,4010) pmname(ipm),now_runs,pmmean,
     > permeas(1),pmlow,pmmedian,pmhigh,permeas(ITERATIONS)
      if (pmmedian .ne. 0) then
       write(iunit,4011)
     >    permeas(1)/pmmedian,pmlow/pmmedian,pmmedian/pmmedian,
     >    pmhigh/pmmedian,permeas(ITERATIONS)/pmmedian
      endif

C     --WRITE HISTOGRAM OF PERFORMANCE MEASURES
4020  format(/t9,'BOOTSTRAP INTERVALS -- ',a//
     >       '    Fraction of Median'/
     >       '   Min. < Middle < Max.     Count')
4022  format(f7.3,' <',f7.2,' <',f8.3,2x,200a1/(28x,200a1))

      if (pmmedian .ne. 0) then
       write(iunit,4020) pmname(ipm)
       fractmin = permeas(1)/pmmedian
       intervals = 2*(1.00-FRACTMAX1)/FRACTINC +2.0005
       fractmax = FRACTMAX1
       tvalmax = pmmedian *fractmax
       ib = 1
       sumpop = 0
       do 600 i=1,intervals
          tvalpop = 0
580         if (ib .le. ITERATIONS) then
             if (permeas(ib) .le. tvalmax) then
              tvalpop = tvalpop +1
              ib = ib +1
              goto 580
             endif
          endif
          write(iunit,4022) fractmin,(fractmin+fractmax)*.5,fractmax,
     >       ('+',j=1,tvalpop)
```

```
          sumpop = sumpop +tvalpop
          fractmin = fractmax
          if (i.lt.(intervals-1)) then
             fractmax = fractmax +FRACTINC
             tvalmax = pmmedian *fractmax
          else
             fractmax = permeas(ITERATIONS)/pmmedian
             tvalmax = permeas(ITERATIONS)
          endif
600      continue
       if (sumpop .ne. ITERATIONS) then
          write(6,'(''WRITENOW error: sumpop .ne. ITERATIONS'')')
          stop 1
       endif
      endif

C     --PRINT WHETHER ENOUGH RUNS
      if (answer.eq.'yes') then
       write(iunit,fmt='(/''yes, '',i3,'' runs are enough''/)')
     >    now_runs
       write(6,fmt='(''yes, '',i3,'' runs are enough'')')
     >    now_runs
      else
       write(iunit,fmt='(/''no, '',i3,'' runs are not enough'')')
     >    now_runs
       write(6,fmt='(''no, '',i3,'' runs are not enough'')')
     >    now_runs
      endif
      return
      end
```

```
C-----------------------------------------------------------------------
      subroutine writesix (ipm,iunit,now_runs,
     > pmmean,pmlow,pmmedian,pmhigh,permeas,ITERATIONS,answer)
C#PARAMETERS
      integer ipm,iunit,now_runs,ITERATIONS
      real pmmean,pmlow,pmmedian,pmhigh,permeas(ITERATIONS)
      character*(*) answer
C#LOCAL
      character*14 pmname(3)
      data pmname/'Exchange Ratio',
     >            'Reds Killed',
     >            'Blues Killed'/
C#ENDDEC
C     --WRITE CONFIDENCE INTERVALS BY BOOTSTRAPPING
4010  format(
     >       '                  CONFIDENCE INTERVALS BY BOOTSTRAPPING'//
     >       '                         <---Extrema of 90%--->        '/
     >       ' Enough                    5th             95th        '/
     >       '  Runs?  Mean      Min     %ile  Median    %ile     Max')
4012  format(
     >       ' ------  ------  ------  ------  ------  ------  ------',
     >       '  ------------------')
4014  format(2x,a3,2x,f8.2,5f8.2,2x,a)
4015  format(7x,     8x,  5f8.3,'  Fraction of Median')

      if (ipm .eq. 1) then
       write(iunit,*) ' '
       write(iunit,4010)
       write(iunit,4012)
      endif
      write(iunit,4014) answer,pmmean,
     > permeas(1),pmlow,pmmedian,pmhigh,permeas(ITERATIONS),
     > pmname(ipm)
      if (pmmedian .ne. 0) then
       write(iunit,4015)
     >    permeas(1)/pmmedian,pmlow/pmmedian,pmmedian/pmmedian,
     >    pmhigh/pmmedian,permeas(ITERATIONS)/pmmedian
      endif
      write(iunit,4012)
      return
      end
C-----------------------------------------------------------------------
```

Appendix C. *STATS*

```
# Purpose:     STATS script to post-process Brawler runs.
# Author(s):   LAWRENCE J. TARANTO
#              JIM WILLIAMS (SVERDRUP)
#              ASC/XRA, BUILDING 16
#              2275 D STREET SUITE 10
#              WRIGHT-PATTERSON AFB OH  45433-7227
# Date:        14 OCTOBER 1997
# Notes:       Compatible with Brawler V6.3
#
# Determine terminal connection
set Terminal=`tty | sed 's:^/dev/::' `
if ( "X$Terminal" == "X" || "X$Terminal" == "Xnot a tty") set Terminal=null
if ("$Terminal" == "null") then
   set Terminal="/dev/null"
else
   set Terminal="/dev/$Terminal"
endif
#
# Set local variables
set No=0
set RunPath="$STATS"
set Yes=1
#
# Initialize
set OverWrite=$Yes
set PreProcess=$Yes
set Prompts=$No
#
# Process arguments
if ("$1" == "") then
   set ManyDirList=""
else if ("$1" == "batch") then
   set ManyDirList=""
   set Prompts=$No
else
   set ManyDirList=$1
endif
#
# Set date and time
set CurrentDate="`date '+%d%b%y'`"
set CurrentTime="`date '+%H%M.%S'`"
#
echo " " > $Terminal
echo "Brawler Post-Processor" > $Terminal
echo "   Execution Date/Time:  $CurrentDate / $CurrentTime" > $Terminal
echo " " > $Terminal
#
# Determine if any "many" directories exist
if ("$ManyDirList" == "") then
   ls -d many* >& /dev/null
   if ($status != 0) then
      # Determine if any LOG and/or database files exist in current directory
      ls LOG.* >& /dev/null
      set LOGstatus=$status
      ls database >& /dev/null
      set DBstatus=$status
      if ($LOGstatus == 0 | $DBstatus == 0) then
         set ManyDirList="."
      else
         echo " "
         echo 'ERROR:  No "many" directories, "LOG" files, or "database" files
found...'
         echo " "
         exit
      endif
   else
      set ManyDirList=`ls -d many*`
   endif
endif
#
if ($Prompts == $Yes) then
   csh -f $RunPath/yesno.csh "If Database Files Exists, Overwrite" $Yes 3
$Terminal
   set OverWrite=$status
   #
   if ($OverWrite == $Yes) then
      csh -f $RunPath/yesno.csh "Pre-Process LOG Files" $Yes 3 $Terminal
      set PreProcess=$status
   else
      set PreProcess=$No
   endif
endif
#
set CurrentDir=`pwd`
foreach ManyDir ($ManyDirList)
   cd $CurrentDir/$ManyDir
   echo " " > $Terminal
   echo '   Processing In Directory "'`basename $cwd`'...' > $Terminal
   #
   ls LOG.* >& /dev/null
   if ($status != 0) then
      echo " "
      echo "ERROR:  No LOG files found..."
   else
      rm -f STATS.prt
      touch STATS.prt
#      echo "Brawler Post-Processor" >> STATS.prt
#      echo "   Execution Date/Time:  $CurrentDate / $CurrentTime" >> STATS.prt
#      echo " " >> STATS.prt
#      echo "   Task:        $Task" >> STATS.prt
#      echo "   Directory:   `basename $cwd`" >> STATS.prt
#      echo " " >> STATS.prt
      #
#      echo "   Task:        $Task" >> $Terminal
#      echo "   Directory:   `basename $cwd`" >> $Terminal
#      echo " " >> $Terminal
      #
      set InputFiles="`ls LOG.* | sort -t. +1n`"
      #
      ls database >& /dev/null
      set DBstatus=$status
      if ($DBstatus == 2 | $OverWrite == $Yes) then
         echo "      Creating Database File..."
         rm -f database
         touch database
         #
         if (-e STATS.dat) then
            nawk -f $RunPath/read_input.awk STATS.dat >> database
         endif
         #
         foreach InputFile ($InputFiles)
            echo -n '         Processing LOG File "'$InputFile'"...'
            rm -f database.tmp
            nawk -v PrePro=$PreProcess -f $RunPath/build_database.awk $InputFile
> database.tmp
            if ($status == 1) then
               echo " (Good)"
               cat database.tmp >> database
            else
               echo " (Bad)"
               set Field2=`echo $InputFile | cut -f2 -d.`
               mv $InputFile log.$Field2
            endif
         end
         rm -f database.tmp
      endif
      #
#      echo "      Generating Aircraft Survivability Report..."
#      nawk  -f $RunPath/ac_summary.awk database >> STATS.prt
      #
##      echo "      Generating Aircraft Kill Report..."
##      /usr/local/bin/perl $RunPath/kill.perl database LOG.1 >> STATS.prt
      #
#      echo "      Generating Mission Success Report..."
#      nawk  -f $RunPath/msn_success.awk database >> STATS.prt
      #
      echo "      Generating Missile Effectiveness Report..."
      nawk -f $RunPath/msl_summary.awk database >> STATS.prt
      #
#      echo "      Generating Missile Launch Report..."
#      nawk -f $RunPath/msl_launch.awk database >> STATS.prt
      #
#      echo "      Generating Missile Launch Distribution Reports..."
#      nawk -f $RunPath/msl_launch_dis_bvr.awk database >> STATS.prt
#      nawk -f $RunPath/msl_launch_dis_wvr.awk database >> STATS.prt
      #
#      echo "      Generating Missile Shot/Kill Breakdown Report..."
#      nawk -f $RunPath/msl_breakdown.awk database >> STATS.prt
      #
#      echo "      Generating Missile Range/Aspect Report..."
#      nawk -f $RunPath/msl_rng_asp.awk database >> STATS.prt
      #
#      echo "      Generating Gun Reports..."
#      nawk -f $RunPath/gun_rng.awk database >> STATS.prt
#      nawk -f $RunPath/gun_aspect.awk database >> STATS.prt
#      nawk -f $RunPath/gun_rng_aspect.awk database >> STATS.prt
#      nawk -f $RunPath/gun_rng_aspect_pk.awk database >> STATS.prt
#      nawk -f $RunPath/gun_time_rng.awk database >> STATS.prt
      #
#      echo "      Generating Weapon Time/Range Report..."
#      nawk -f $RunPath/wpn_time_rng.awk database >> STATS.prt
      #
   endif
end
#
exit
```

Appendix D. *STATS.prt* Missile Effectiveness Reports

```
MISSILE EFFECTIVENESS REPORT
===========================================================

                          MSLI         MSLR
                          ------------ ------------
Firings                      537         1759
Fuzings                      294         1108
Kills                        172          626
Kills/Firing                 32%          35%

BLUE:
   RF Msl Firings/Kills           1759          626
   IR Msl Firings/Kills            483          165
RED:
   RF Msl Firings/Kills              0            0
   IR Msl Firings/Kills             54            7

THUNDER MISSILE EFFECTIVENESS REPORT
============================================================

                          MSLI         MSLR
                          ------------ ------------
Adj. Kills/Firing           0.32         0.36
```

Appendix E. *msl_summary.awk*

```
# Author:      LAWRENCE J. TARANTO
#              ASC/XRA, BUILDING 16
#              2275 D STREET SUITE 10
#              WRIGHT-PATTERSON AFB OH  45433-7227
# Purpose:     To generate missile summary report.
# Date:        15 OCTOBER 1997
#
BEGIN {
FS="|"
   cases=0
   b_total=0
   r_total=0 }
#
{
   rec_type=$2
   if (rec_type == "MSL") {
      mslid=$3
      mstyp=$4
      launcher[mstyp]=$5
      freason=$10
      fuzee=$11
      kill=$12
      if (kill == "T") {kill=1}
      if (kill == "F") {kill=0}
      #
      if (freason != "") {
         failure[freason,mstyp]++
         fmode[freason]++
         failed[mstyp]++
         if (fuzee != 0) { fuz_n_fail[freason,mstyp]++ }
      }
      #
      mode=$14
      launch_mode[mstyp,mode]++
      if (kill == 1) { lethal_launch_mode[mstyp,mode]++ }
      b_init=$15
      shooter=int(substr(mslid, 1, 2))
      if (shooter > b_init) { blue_red=2 }
      if (shooter <= b_init) { blue_red=1 }
      if (mode == "SEMI_ACTIVE_SEEKER_LOCK") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "CMD_GUIDED_DES_RDR") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "CMD_GUIDED_DES_ITB") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "CMD_GUIDED_DES_OTD") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "CMD_GUIDED_DES_STT") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "CMD_GUIDED_UNDES_VIS") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "CMD_GUIDED_UNDES_IRST") {
         modes[mode]++
         rf_fires[blue_red]++
         if (kill == 1) { rf_kills[blue_red]++ } }
      else if (mode == "PASSIVE_SEEKER_LOCK_IR") {
         modes[mode]++
         ir_fires[blue_red]++
         if (kill == 1) { ir_kills[blue_red]++ } }
      else if (mode == "LOCK_ON_AFTER_LN_HMS") {
         modes[mode]++
         ir_fires[blue_red]++
         if (kill == 1) { ir_kills[blue_red]++ } }
      else if (mode == "LOCK_ON_AFTER_LN_RDR") {
         modes[mode]++
         ir_fires[blue_red]++
         if (kill == 1) { ir_kills[blue_red]++ } }
      else {
         printf("\n%s\n",">>> WARNING:  UNKNOWN MISSILE LAUNCH MODE
ENCOUNTERED <<<") }
      #
      launchings[mstyp]++
      if (freason == " FUZED ON DEAD TARGET ") { fuzed_dead[mstyp]++ }
      if (fuzee != 0) { fuzed[mstyp]++ }
      if (kill == 1) { killed[mstyp]++ }
   }
   if (rec_type == "AC ") {
      cases=cases+$3
      b_total=b_total+$5
      r_total=r_total+$11
   }
}
#
END {
#
#printf "\n\n%27s", "MISSILE EFFECTIVENESS REPORT\n"
#printf "%100s",
"=======================================================================
=============================\n\n"
#
#printf("%25s"," ")
#for (i in launchings) { printf("%13s",i) }
#printf("\n%25s"," ")
#for (i in launchings) { printf(" ------------") }
#printf("\n%-19s","Firings   ")
#for (i in launchings) { printf("%13d",launchings[i]) }
#printf("\n%-19s","Fuzings   ")
#for (i in launchings){ printf("%13d",fuzed[i]) }
#printf("\n%-19s","Kills     ")
#for (i in launchings) { printf("%13d",killed[i]) }
#printf("\n%-25s","Fuzings/Firing")
for (i in launchings) {
   rate=100.0*(fuzed[i]/launchings[i])
#   printf("%7d%%%5s",rate," ")
```

```
}
#printf("\n%-25s","Kills/Firing  ")
for (i in launchings) {
   rate=100.0*(killed[i]/launchings[i])
#   printf("%7d%%%5s",rate," ")
}
#printf("\n%-25s","Kills/Fuzing  ")
for (i in launchings) {
   rate=0.0
   if (fuzed[i]!=0) rate=100.0*(killed[i]/fuzed[i])
#   printf("%7d%%%5s",rate," ")
 }
#printf("\n")
#printf("\n%-5s","BLUE:")
#printf("\n%-25s","   RF Msl Firings/Kills")
#printf("%13d%13d",rf_fires[1],rf_kills[1])
#printf("\n%-25s","   IR Msl Firings/Kills")
#printf("%13d%13d",ir_fires[1],ir_kills[1])
#printf("\n%-4s","RED:")
#printf("\n%-25s","   RF Msl Firings/Kills")
#printf("%13d%13d",rf_fires[2],rf_kills[2])
#printf("\n%-25s","   IR Msl Firings/Kills")
#printf("%13d%13d",ir_fires[2],ir_kills[2])

#
#printf "\n\n\n%36s", "THUNDER MISSILE EFFECTIVENESS REPORT\n"
#printf "%100s",
"=======================================================================
=============================\n\n"
#
#printf("%25s"," ")
#for (i in launchings) { printf("%13s",i) }
#printf("\n%25s"," ")
#for (i in launchings) { printf(" ------------") }
printf("\n%-25s","Adj. Kills/Firing  ")
for (i in launchings) {
   if (fuzed[i] > 0) {
      den=launchings[i]-(fuzed_dead[i]*(launchings[i]/fuzed[i]))
      rate=killed[i]/den }
   else {rate=0.0}
      printf("%7.2f%6s",rate," ")
}
#
printf("\n\n")
b_fires=0
r_fires=0
for (i in launchings) {
   if (launcher[i] <= b_total/cases) { b_fires=b_fires+launchings[i] }
   if (launcher[i] >  b_total/cases) { r_fires=r_fires+launchings[i] }
}
#printf "%25s %6.2f\n","BLUE Shots/Aircraft:", b_fires/b_total
#printf "%25s %6.2f","RED  Shots/Aircraft:", r_fires/r_total
#
printf "\n\n\n%23s", "MISSILE FAILURES REPORT\n"
printf "%100s",
"=======================================================================
=============================\n\n"
#
printf("%-25s"," FAILURE MODE")
for (i in launchings) { printf("%13s",i) }
printf("\n%25s"," ")
```

```
for (i in launchings) { printf(" ------------") }
printf("\n")
for (i in fmode) {
   printf("%-25s",i)
   for (l in launchings) {
      row[l,1]=failure[i,l]
      row[l,2]=fuz_n_fail[i,l]
      sum[l,1]+=failure[i,l]
      sum[l,2]+=fuz_n_fail[i,l] }
      for(l in launchings) { printf("%7d/%5d",row[l,1],row[l,2]) }
      printf("\n") }
#
printf("\n%-25s"," FAILED BUT FUZED")
for (l in launchings) { printf("%7d%6s",sum[l,2]," ") }
printf("\n%-25s"," HARD FAILURES")
for (l in launchings) { printf("%7d%6s",sum[l,1]-sum[l,2]," ") }
#
printf "\n\n\n%26s", "MISSILE LAUNCH MODE REPORT\n"
printf "%100s",
"=======================================================================
=============================\n\n"
#
printf("%-25s"," LAUNCH MODE")
for (i in launchings) { printf("%13s",i) }
printf("\n%25s"," ")
for (i in launchings) { printf(" ------------") }
printf("\n")
for (i in modes) {
   printf("%-25s",i)
   for (l in launchings) { printf("%7d%6s",launch_mode[l,i]," ") }
   printf("\n") }
#
printf "\n\n\n%26s", "MISSILE LETHAL LAUNCH MODE REPORT\n"
printf "%100s",
"=======================================================================
=============================\n\n"
#
printf("%-25s"," LETHAL LAUNCH MODE")
for (i in launchings) { printf("%13s",i) }
printf("\n%25s"," ")
for (i in launchings) { printf(" ------------") }
printf("\n")
for (i in modes) {
   printf("%-25s",i)
   for (l in launchings) { printf("%7d%6s",lethal_launch_mode[l,i]," ")
}
   printf("\n") } }
}
```

Appendix F. *make_flyAKPF*

```
# C-shell program make_flyAKPF
#
# Author:  Bryan Livergood
#
# This program searches all BRAWLER log files in a directory
# and obtains Firings and Adjusted Kills per Firing (AKPF)
# data for each log.  It then creates one file called
fly.FAKPF
# that contains a list of all the Firings values followed by
a
#list of all the AKPF values for each BRAWLER run.
#
#!/bin/csh
mkdir akpfSTATS
foreach fyle (`ls LOG.* $1`)
  echo 'processing '$fyle
  grep ALIVE $fyle
   echo 'grep status '$status
   if (!($status)) then
     set fylenum = `ls $fyle | cut -dG -f2`
     echo 'log number '$fylenum
     mkdir temporary
     cp $fyle ./temporary
     cd temporary
     STATS
     grep -h 'Firings' STATS.prt >> ../fly.Fire+AKPF
     grep -h 'Adj. Kills/Firing' STATS.prt >>
../fly.Fire+AKPF
     mv STATS.prt ../akpfSTATS/STATS$fylenum
     rm *
     cd ..
     rmdir temporary
   endif
end
grep -h 'Adj. Kills/Firing' fly.Fire+AKPF > fly.FAKPF
grep -h 'Firings    ' fly.Fire+AKPF >> fly.FAKPF
```

Appendix G. BRAWLER AKPF and Firings Data

Replication	Scenario 1		Scenario 2		Scenario 3	
Number	AKPF	Firings	AKPF	Firings	AKPF	Firings
1	0.40	5	0.00	6	0.0	0
2	0.00	4	0.00	0	0.0	2
3	0.25	4	0.00	5	0.4	5
4	0.00	1	0.00	0	0.5	2
5	0.50	4	0.67	3	0.7	3
6	0.20	5	0.12	8	1.0	1
7	0.25	4	0.33	3	0.0	3
8	0.50	4	0.50	2	0.0	1
9	0.00	2	0.67	3	0.2	6
10	1.00	1	0.25	4	0.7	3
11	0.00	0	0.50	2	0.0	0
12	0.00	2	0.33	3	0.5	2
13	0.00	1	0.00	1	0.0	0
14	0.20	5	0.00	2	0.3	3
15	0.00	0	0.40	5	0.5	2
16	0.00	2	0.50	2	0.0	4
17	0.50	2	1.00	2	0.3	4
18	0.40	5	0.00	4	0.0	2
19	0.00	0	0.00	2	0.0	1
20	0.33	3	0.00	3	1.0	2
21	1.00	2	1.00	1	1.0	1
22	1.00	1	0.20	5	0.5	2
23	0.00	1	0.00	2	0.0	0
24	0.00	0	0.25	4	0.0	0
25	0.67	3	0.67	3	0.5	2
26	1.00	2	0.00	3	0.7	3
27	0.00	5	0.33	3	0.0	2
28	1.00	1	0.67	3	0.0	1
29	0.50	2	0.00	2	0.0	0
30	0.00	2	0.00	5	0.0	1
31	0.00	0	0.67	3	1.0	2
32	0.00	3	0.25	4	0.0	0
33	0.00	0	0.33	3	0.0	2
34	0.25	4	1.00	1	0.5	2
35	0.00	0	0.33	3	0.0	2
36	0.00	3	0.33	3	0.0	0
37	0.00	4	0.50	2	1.0	1
38	1.00	1	0.00	1	1.0	2
39	0.33	3	1.00	1	0.0	0
40	1.00	1	0.17	6	1.0	1

Replication	Scenario 1		Scenario 2		Scenario 3	
Number	AKPF	Firings	AKPF	Firings	AKPF	Firings
41	0.33	3	0.00	0	0.5	4
42	0.20	5	0.00	0	0.0	4
43	0.00	3	0.00	1	0.5	2
44	1.00	1	0.00	4	0.0	4
45	0.00	1	0.00	1	0.5	2
46	0.17	6	1.00	1	0.0	2
47	0.00	1	0.25	4	0.3	3
48	0.00	0	0.50	2	1.0	1
49	0.17	6	1.00	1	0.0	0
50	0.50	2	0.50	2	0.3	6
51	0.17	6	0.00	0	1.0	1
52	0.00	0	0.33	3	0.0	1
53	0.20	5	0.17	6	0.3	3
54	0.00	0	0.00	0	0.0	2
55	0.33	3	0.00	1	0.0	0
56	0.50	4	0.33	3	0.7	3
57	0.00	1	0.50	2	0.5	2
58	0.50	4	0.50	2	0.0	8
59	1.00	2	1.00	1	0.0	0
60	0.00	3	0.00	2	0.0	1
61	0.00	2	0.00	3	1.0	1
62	0.00	0	0.50	4	0.0	1
63	0.00	0	0.00	0	0.3	3
64	0.50	2	0.00	3	0.0	4
65	0.00	0	0.50	4	0.0	2
66	1.00	2	0.00	2	1.0	2
67	0.00	0	0.00	2	0.1	7
68	1.00	2	0.00	3	0.0	4
69	0.00	3	0.00	3	0.0	1
70	0.00	4	0.40	5	0.0	0
71	0.50	2	0.00	0	0.0	1
72	0.20	5	1.00	1	0.0	0
73	0.00	2	0.20	5	0.0	1
74	1.00	1	0.00	3	0.0	0
75	0.40	5	0.25	4	0.0	2
76	0.00	2	0.00	0	0.0	1
77	0.25	4	0.00	1	1.0	1
78	1.00	1	0.50	2	0.0	2
79	1.00	1	0.00	5	0.0	1
80	0.00	0	0.00	1	0.0	1

Replication	Scenario 1		Scenario 2		Scenario 3	
Number	AKPF	Firings	AKPF	Firings	AKPF	Firings
81	0.00	0	0.00	1	0.5	2
82	0.33	3	0.43	7	0.0	1
83	1.00	1	0.40	5	1.0	1
84	0.00	1	0.50	2	0.5	2
85	0.00	2	0.00	3	0.4	5
86	0.33	3	1.00	3	0.0	1
87	0.50	2	0.00	2	0.0	0
88	0.50	4	0.50	2	0.5	2
89	1.00	1	0.00	3	0.0	3
90	0.00	0	0.00	0	0.2	6
91	1.00	2	0.00	0	0.0	1
92	1.00	1	0.00	1	0.0	1
93	0.00	0	0.33	3	0.3	4
94	1.00	1	0.17	6	0.0	1
95	0.17	6	0.00	2	0.3	3
96	1.00	2	0.50	2	0.0	0
97	0.14	7	0.33	3	0.0	0
98	0.50	2	0.20	5	0.5	4
99	0.00	0	0.00	2	0.3	3
100	0.67	3	0.33	3	0.0	1
101	0.00	0	0.50	2	0.0	0
102	1.00	1	1.00	1	0.0	1
103	0.00	4	0.33	3	0.0	4
104	0.00	2	0.00	1	0.0	2
105	0.33	3	0.25	4	1.0	1
106	0.14	7	0.33	3	0.3	3
107	0.00	5	0.00	5	0.0	6
108	0.00	0	0.50	2	0.0	0
109	0.00	2	1.00	2	1.0	1
110	0.00	0	0.00	1	1.0	1
111	0.50	2	0.00	0	1.0	1
112	0.00	2	0.00	4	0.5	4
113	0.00	0	0.40	5	0.3	3
114	0.50	4	1.00	1	0.0	0
115	0.00	1	0.25	4	0.4	5
116	0.00	0	0.33	3	0.5	2
117	0.00	3	0.50	4	0.0	1
118	0.50	2	0.25	4	0.0	3
119	0.00	0	0.20	5	0.3	3
120	0.00	0	0.00	4	1.0	1

Replication	Scenario 1		Scenario 2		Scenario 3	
Number	AKPF	Firings	AKPF	Firings	AKPF	Firings
121	0.33	3	1.00	1	0.0	1
122	0.50	4	0.00	2	0.1	7
123	0.10	10	0.50	2	0.2	5
124	0.33	3	0.25	4	0.0	0
125	0.00	0	1.00	1	0.0	0
126	0.00	0	0.00	2	0.1	9
127	0.00	4	0.50	2	0.0	0
128	0.00	7	0.67	3	0.3	4
129	0.33	3	0.33	3	1.0	1
130	0.00	0	0.00	2	0.0	0
131	0.33	6	0.00	1	0.0	1
132	1.00	1	0.33	3	0.0	1
133	0.00	2	0.25	4	0.0	0
134	0.00	1	0.00	1	1.0	1
135	0.50	2	0.40	5	0.3	3
136	0.25	4	0.60	5	0.0	2
137	0.20	5	0.33	3	0.0	2
138	1.00	3	0.67	3	0.0	0
139	0.67	3	1.00	1	0.0	0
140	0.00	1	0.00	1	0.0	0
141	0.00	1	0.00	1	0.0	0
142	0.00	0	0.25	4	0.0	0
143	0.00	3	0.00	0	0.0	2
144	0.50	2	0.33	3	0.0	0
145	0.00	0	0.33	3	0.0	4
146	0.25	4	1.00	3	1.0	1
147	1.00	1	0.33	3	0.0	0
148	0.50	2	0.50	4	0.3	3
149	0.50	2	0.00	2	0.4	5
150	0.33	3	0.00	0	0.3	4
151	0.00	0	0.33	3	0.0	0
152	0.00	0	0.00	0	0.0	0
153	0.50	2	1.00	1	0.3	3
154	0.25	8	0.50	2	0.3	6
155	1.00	1	1.00	1	0.0	0
156	0.33	3	1.00	1	0.5	2
157	0.00	0	0.50	2	0.0	0
158	0.00	1	0.50	2	0.0	0
159	0.00	1	0.67	3	1.0	2
160	0.75	4	0.20	5	0.0	3

Replication	Scenario 1		Scenario 2		Scenario 3	
Number	AKPF	Firings	AKPF	Firings	AKPF	Firings
161	0.00	3	1.00	1	0.0	3
162	0.00	0	0.00	2	0.5	4
163	0.00	0	1.00	1	0.0	3
164	0.00	1	1.00	1	0.0	0
165	0.00	3	0.00	1	0.0	0
166	1.00	1	0.00	3	0.0	1
167	0.50	2	0.40	5	0.3	3
168	0.00	1	0.50	2	0.0	1
169	0.50	2	0.00	0	0.4	5
170	1.00	1	1.00	1	0.0	2
171	0.50	4	0.00	4	1.0	1
172	0.00	1	0.00	0	0.0	2
173	0.50	2	0.50	2	0.0	3
174	0.00	2	0.67	3	0.0	0
175	0.00	0	0.25	4	1.0	1
176	1.00	1	0.50	4	0.0	0
177	0.40	5	0.50	2	0.0	0
178	0.43	7	0.00	0	0.0	1
179	0.00	3	0.33	3	0.5	4
180	0.00	2	0.50	2	1.0	1
181	0.14	7	0.00	4	0.5	2
182	0.50	6	0.00	0	0.3	4
183	0.00	0	0.50	6	0.0	2
184	0.33	3	0.67	3	0.0	1
185	0.00	1	0.00	4	0.5	2
186	0.50	2	0.67	3	0.0	1
187	0.50	4	0.50	2	0.3	3
188	0.00	2	0.00	2	0.0	0
189	0.50	2	0.33	3	0.0	1
190	0.00	1	1.00	1	0.0	0
191	0.00	1	0.00	3	0.5	2
192	0.00	1	0.60	5	0.0	0
193	1.00	2	0.00	1	0.0	2
194	0.33	3	*	*	0.0	2
195	0.33	3	*	*	0.0	0
196	0.33	3	*	*	0.0	0
197	0.50	2	*	*	0.5	2
198	0.00	1	*	*	0.3	6
199	0.33	3	*	*	0.0	0
200	0.67	3	*	*	0.0	1

Appendix H. Derivation of Adjusted Kills per Firing (AKPF)

The following equations are presented in Section 4.4.1 and are presented again here:

$$KPF = \frac{\sum Kills}{\sum Firings} \tag{4.2}$$

$$AKPF = \frac{\sum Kills}{\sum Firings - \frac{\sum Fuzings_{DEAD} \cdot \sum Firings}{\sum Fuzings}} \, . \tag{4.3}$$

$$AKPF = KPF \cdot \frac{\sum Fuzings}{\left(\sum Fuzings - \sum Fuzings_{DEAD}\right)} \tag{4.4}$$

These are the equations that STATS uses to calculate kills per firing (KPF) and adjusted kills per firing (AKPF).

We now present the derivation of (4.4) from (4.3). The actual computation performed by *STATS* follows:

$$AKPF = \frac{\sum Kills}{den}$$

where

$$den = \sum Firings - \frac{\sum Fuzings_{DEAD} \cdot \sum Firings}{\sum Fuzings}$$

which we manipulate as follows:

$$den = \frac{\sum Firings \cdot \sum Fuzings}{\sum Fuzings} - \frac{\sum Fuzings_{DEAD} \cdot \sum Firings}{\sum Fuzings}$$

to obtain:

$$den = \frac{\sum Firings \cdot \left(\sum Fuzings - \sum Fuzings_{DEAD}\right)}{\sum Fuzings}$$

Since

$$AKPF = \frac{\sum Kills}{den} \quad \Rightarrow \quad AKPF = \sum Kills \cdot \frac{1}{den}$$

This implies:

$$\begin{aligned} AKPF &= \sum Kills \cdot \frac{\sum Fuzings}{\sum Firings \cdot \left(\sum Fuzings - \sum Fuzings_{DEAD}\right)} \\ &= \frac{\sum Kills}{\sum Firings \cdot} \cdot \frac{\sum Fuzings}{\left(\sum Fuzings - \sum Fuzings_{DEAD}\right)} \end{aligned}$$

which gives the desired result:

$$AKPF = KPF \cdot \frac{\sum Fuzings}{\left(\sum Fuzings - \sum Fuzings_{DEAD}\right)} \tag{4.4}$$

Bibliography

1. Aeronautical Systems Center Development Planning Directorate, Campaign Analysis (ASC/XRA). "Baseline Database Pedigree, Appendix Air: Air-to-Air Pedigree, Version 2.0 (Unclassified)." February 1996.

2. Aeronautical Systems Center Development Planning Directorate; Modeling, Simulation and Analysis, (ASC/XRB). "Concept of Operations: Modeling, Simulation and Analysis Focus Area." 1996.

3. Aeronautical Systems Center Simulation and Analysis Facility (ASC/SM). "What is SIMAF." Briefing. June 1997.

4. Air Force Studies and Analysis Agency (AFSAA). THUNDER Analyst's Manual Version 6.4. Arlington VA: CACI Products Company. 1995.

5. Beene, Eric, Jonathon Clough, and Scott Fox. "Comparison of Two Missiles: An Analysis of BRAWLER Output." Report to Air Force Institute of Technology Combat Modeling Class. Air Force Institute of Technology, Wright-Patterson AFB OH, September 1997.

6. The BRAWLER Air Combat Simulation Analyst Manual (Rev 6.2 -- Draft), Technical Report 906, Decision Science Applications, Inc., August 1995

7. The BRAWLER Air Combat Simulation User Manual – Part I (Rev 6.2 -- Draft), Technical Report 908, Decision Science Applications, Inc., October 1995.

8. Bundy, Gary N., David W. Seidel, Ben C. King, and Carl D. Burke. "An Experiment in Simulation Interoperability," Proceedings of the 1996 Winter Simulation Conference (1996).

9. Buschor, Daniel C. Sensitivity Analysis of BRAWLER Pilot Skill Levels. MS thesis, AFIT/GOA/ENS/98M-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1998.

10. Cheng, Russell C. H. "Bootstrap Methods in Computer Simulation Experiments," Proceedings of the 1995 Winter Simulation Conference, 171-177 (1995).

11. Davis, Paul K. "An Introduction to Variable-Resolution Modeling," Naval Research Logistics, 42:2, 151-181 (March 1995).

12. DiCiccio, Thomas J. and Bradley Efron. "Bootstrap Confidence Intervals," Statistical Science, 11:3: 189-228 (1996).

13. Efron, Bradley. "Bootstrap Methods: Another Look at the Jackknife," Annals of Statistics, 7, 1-26 (1979).

14. -----. The Jackknife, the Bootstrap, and Other Resampling Plans, Philadelphia PA: Society for Industrial and Applied Mathematics, 1982.

15. Efron, Bradley, and Tibshirani, R. J. An Introduction to the Bootstrap, New York NY: Chapman & Hall, Inc., 1993.

16. Farmer, Michael R., The Effects of Changing Force Structure on THUNDER Output.. MS Thesis, AFIT/GOA/ENS/96M-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1996.

17. Friedman, Linda W. and Hershey H. Friedman. "Analyzing Simulation Output Using the Bootstrap Method," Simulation 64:2, 95-100,Simulation Councils, Inc. (February 1995).

18. Grier, James B. Linking Procurement Dollars to an Alternative Force Structures' Combat Capability Using Response Surface Methodology. MS Thesis, AFIT/GOA/ENS/97M-07. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1997.

19. Hall, Peter. The Bootstrap and Edgeworth Expansion, New York NY: Springer-Verlag, 1992.

20. Hardy, Doug and Mike Healy. "Constructive & Virtual Interoperation: A Technical Challenge," Proceedings of the Military, Government, and Aerospace Simulation Conference, 37-42 (1994).

21. Hillestad, Richard, John Owen, and Don Blumenthal, "Experiments in Variable Resolution Combat Modeling," Naval Research Logistics 42:2, 209-232 (March 1995).

22. Hillestad, Richard, and Louis Moore, The Theater-Level Campaign Model: A Research Prototype for a New Generation of Combat Analysis Model. RAND, 1996 (MR-388-AF/A).

23. Johnson, Carol, RAND Corporation. "How Many TAC Brawler Runs are Enough?" Report to the BRAWLER User Group. November 1994.

24. Kim, Wonsik. Personnel Airdrop Risk Assessment Using Bootstrap Sampling. MS Thesis, AFIT/GOR/ENS/96D-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1996.

25. Langbehn, Skip, Headquarters USAF/XOCA, "Air Force Analysis Toolkit: Legacy Model Transition Plan." Briefing. January 1998.

26. Law, Averill M., and W. David Kelton, Simulation Modeling & Analysis, Second Edition, New York NY: McGraw-Hill, Inc., 1991.

27. Logan, Tammy. Member of the THUNDER analysis group, ASC/XRA, Wright-Patterson AFB OH. Personal and telephone interviews. September 1997-February 1998.

28. Ripley, Brian D., Stochastic Simulation, New York NY: John Wiley & Sons, Inc., 1992.

29. Taranto, Lawrence J., Aeronautical Systems Center Development Planning Directorate, Campaign Analysis (ASC/XRA), "BRAWLER/THUNDER Calibration Methodology." Report to the BRAWLER User Group. 12 December 1995.

30. Taranto, Lawrence J. Member of the BRAWLER analysis group, ASC/XRA, Wright-Patterson AFB, OH. Personal and telephone interviews, October 1997-February 1998.

31. Thompson, David. Member of the THUNDER analysis group, ASC/XRA, Wright-Patterson AFB, OH. Telephone interviews. December 1997-January 1998.

34. Wackerly, Mendenhall, and Scheaffer. Mathematical Statistics with Applications, 5th Edition, Belmont CA: Duxbury Press, 1996

35. Webb, Timothy S. Analysis of THUNDER Combat Simulation Model, MS Thesis, AFIT/GOA/ENS/94M-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1994.

36. Williams, Jim. Member of the BRAWLER analysis group, ASC/XRA, Wright-Patterson AFB, OH. Telephone interview. January 1998.

Vita

Capt Bryan S. Livergood was born on 18 March 1969 in Sioux Falls, South Dakota, the son of Maj Joseph H. (USAF Ret) and Shirley M. Livergood. He graduated from Lewis-Palmer High School in Monument, Colorado, in May 1986. In May 1991, he received his Bachelor of Science Degree in Mathematics from Colorado State University and was commissioned in the USAF. In June 1992 he reported to the Low Altitude Navigation and Targeting Infrared System for Night (LANTIRN) System Program Office, Wright Patterson Air Force Base (WPAFB), Ohio, where he served as an advanced systems project manager in the LANTIRN product improvement branch. In October 1995 he reported to the Joint Strike Fighter (JSF) Support Office, WPAFB, Ohio, where he served as an acquisition project officer. He served there until entering the Air Force Institute of Technology (AFIT), School of Engineering, WPAFB, Ohio, in August 1996, where he attended the AFIT Graduate Program in Operations Research in 1996. Upon graduation from AFIT in March 1998, he was assigned to Headquarters Air Force Operational Test and Evaluation Center, Studies and Analysis, Logistics (HQ AFOTEC/SAL),

Permanent Address: 18931 Birchwood Way
Monument, CO 80132

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 1998

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
CROSS-RESOLUTION COMBAT MODEL CALIBRATION USING BOOTSTRAP SAMPLING

5. FUNDING NUMBERS

6. AUTHOR(S)

Bryan S. Livergood, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
2950 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/98M-16

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Mr Terry Smith
ASC/SM, Bldg 28
2145 Monahan Way
WPAFB, OH 45433-7017

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Kenneth E. Bauer
AFIT/ENS, Bldg 640, 2950 P Street
WPAFB, OH 45433-7765

Professor / Advisor
Department of Operational Sciences
E-mail: kbauer@afit.af.mil
(937) 255-6565 x4326

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The US Air Force uses many combat simulation models to assist them in performing combat analyses. BRAWLER is a high resolution air-to-air combat simulation model used for engagement-level analyses of few-on-few air combat. THUNDER is a low resolution combat simulation model used for campaign-level analyses of theater-level warfare. BRAWLER is frequently used to ensure that THUNDER air-to-air inputs are valid. This thesis describes the confederation of THUNDER and BRAWLER by clearly showing how one particular BRAWLER output, the effectiveness of a missile type, is transformed into THUNDER air-to-air input data. Since BRAWLER is a stochastic simulation model, it is necessary to replicate a number of BRAWLER simulation runs in order to obtain a sufficiently accurate estimate of the mean missile effectiveness, a number which varies for each different BRAWLER combat scenario. This thesis focuses on using two different sequential methods to determine when the minimum number of BRAWLER runs have been performed to obtain a specified relative precision. One method uses classical statistical analysis techniques, while the other uses the more modern technique of bootstrap resampling. The performance of these two methods is compared.

14. SUBJECT TERMS
Simulation, Combat Model, Cross-Resolution Modeling, Model Resolution, Statistical Analysis, Bootstrap, Resampling

15. NUMBER OF PAGES
149

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL